VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Generování vzorů formulí pro skolemizaci
Pattern Generator of Formulas for Skolemization

2012 Marek Janáček

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání diplomové práce

Student: **Bc. Marek Janáček**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2612T025 Informatika a výpočetní technika

Téma: Generování vzorů formulí pro skolemizaci
Pattern Generator of Formulas for Scolemization

Zásady pro vypracování:

Cílem práce je vytvořit vlastní návrh a implementaci systému v jazyce C# pro podporu výuky předmětu Matematická logika, který bude umožňovat:

1. Vyhodnocení vstupu, kterým je formule, jako výstup vydá její složitost podle typu úprav, které musí být provedeny v rámci metody skolemizace.
2. Vygenerování vzoru na základě zadaného řetězce složitosti. Z tohoto vzoru je pak možné generování konkrétního zadání (konkrétní formule pro skolemizaci), přičemž toto zadání bude stejné obtížnosti jako vzor.
3. Srovnání s podobnými systémy pro výuku PL1, pokud v současné době existují.

Práce bude obsahovat:
1. Vymezení jazyka formulí (podmnožiny PL1).
2. Analýzu obtížnosti úloh na skolemizaci.
3. Určení konkrétní podoby řetězce složitosti, jehož jednotlivé části explicitně a strukturovaně zachycují obtížnost příslušné úlohy.
4. Analýzu, návrh a implementaci algoritmů vyhodnocujících na základě zadané formule její konkrétní složitost.
5. Analýzu, návrh a implementaci algoritmů generujících vzory formulí na základě řetězce obtížnosti.
6. Analýzu, návrh a implementaci generování konkrétních úloh ze vzorů.

Seznam doporučené odborné literatury:

Podle pokynů vedoucího diplomové práce.

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Mgr. Martina Číhalová**

Datum zadání: 18.11.2011

Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této diplomové práce dle požadavku čl. 26, odst. 9 Studijního a zkušebního řádu pro studium v magisterských programech VŠB-TU Ostrava.

V Ostravě dne 3. května 2012 ……Janáček……

podpis

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne 3. května 2012 ……Janáček……

podpis

Rád bych tímto poděkoval Mgr. Martině Číhalové a Mgr. Marku Menšíkovi Ph.D. za jejich čas, informace, ochotu, pomoc a cenné rady při řešení této diplomové práce.

Abstrakt

Obsahem této diplomové práce je analýza úloh na skolemizace formulí jazyka predikátové logiky prvního řádu z hlediska obtížnosti pro studenty. Obtížnost úlohy je explicitně a strukturovaně zachycena pomocí řetězce složitosti. Dále se práce zabývá analýzou, návrhem a implementací systému, který na vstupu obdrží formuli PL1 a na výstupu vrátí řetězec složitosti příslušné formule. Posledním a nejdůležitějším předmětem této práce je analýza, návrh s implementace systému, který na základě řetězce složitosti generuje vzory formulí PL1. Systém rovněž umožnuje ze vzorů vytvářet konkrétní úlohy, čili jej bude možné použít při tvorbě příkladů na cvičení a zadání zápočtových písemek v kurzu Matematická logika.

Klíčová slova

Predikátová logika, skolemizace, řetězec složitosti, vzor, formule, gramatika, syntaktický strom, literál, predikát, kvantifikátor, spojka, negace, konjunkce, disjunkce, implikace, ekvivalence, proměnná, funkce, konstanta, hierarchický řád, syntaktická analýza, lexikální analýza, generování.

Abstract

The content of this thesis is the analysis of skolemization of formulas of first order predicate logic language in terms of difficulty for students. Difficulty of the task is structured and explicitly described by the string of complexity. The thesis deals also with the analysis, design and implementation of a system that is given a PL1 formula and returns the string of complexity for that formula. The last and most important object of this thesis is the analysis, design and implementation of a system that generates formula patterns based on given string of complexity. The system also allows you to create concrete tasks, so it can be used to create examples for exercising and exams in course Mathematical Logic.

Keywords

Predicate logic, skolemization, string of complexity ,pattern, formula, grammar, syntactic tree, literal, predicate, quantifier, junction, negation, conjunction, disjunction, implication, equivalence, variable, function, constant, hierarchical order, parsing, lexical analysis, generation.

Seznam použitých symbolů a zkratek

HTML	– Hypertext markup language
PDF	– Portable document format
PL1	– Predikátová logika prvního řádu
SŘBD	– Systém řízení báze dat

Obsah

1 Úvod

Cílem první části této diplomové práce je analýza úloh na převedení do Skolemovy klauzulární formy (dále jen skolemizace) formulí predikátové logiky prvního řádu z hlediska obtížnosti pro studenty. Obtížnost úlohy se pokusíme explicitně a strukturovaně zachytit pomocí tzv. řetězce složitosti. Skolemizace je významná především v kontextu obecné rezoluční metody, kterou lze aplikovat pouze na formule ve Skolemově klauzulární formě. Skolemizace patří mezi nejobtížnější látku probíranou v kurzu Matematická logika.

V další části se budeme zabývat analýzou a návrhem systému, který pro formuli PL1 na vstupu vrátí řetězec složitosti na výstupu. Tento systém bude umožňovat vizualizaci a procvičování této problematiky studenty, čili najde uplatnění zejména jako pomůcka při výuce PL1. V době psaní této práce žádný jiný podobný systém, který je veřejně dostupný, neexistuje. Obsahem práce je rovněž implementace tohoto systému na platformě .NET v programovacím jazyce C#.

V poslední části se budeme zabývat analýzou, návrhem, a nakonec implementací systému, který pro řetězec složitosti na vstupu vrátí vzory formulí na výstupu. Ze vzorů bude systém schopen generovat konkrétní úlohy. Možnost vytvářet úlohy na základě řetězce složitosti je velice žádoucí při tvorbě písemek, a také při procvičování skolemizace. Pedagog předem ví, které dílčí kroky bude muset student během skolemizace použít a může tak nechat systém vytvořit příklady k procvičení konkrétních kroků, nebo vytvořit více variant písemek se stejnou obtížností. Žádný jiný veřejně dostupný systém v době psaní této práce neexistuje.

2 Teorie

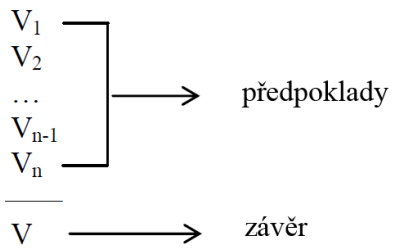
Chceme-li analyzovat a řešit problém z konkrétní oblasti, musíme nejprve definovat základní pojmy, které budeme používat. Proto v následujících podkapitolách shrneme nejnutnější definice z oblasti logiky, a také formální teorie jazyků, které přímo souvisejí s obsahem této diplomové práce a jsou nutné pro její plné pochopení.

2.1 Logika

V případě všech definic v podkapitole Logika vycházím z [1].

Logika je formální věda zabývající se především správným usuzováním a uměním správné argumentace. Jinak řečeno logika zkoumá vztahy (relace) mezi předpoklady (premisami) a závěry. V této práci se budeme zabývat především relací vyplývání.

Obecně můžeme úsudek charakterizovat následovně:
Na základě pravdivosti výroků (soudů, tvrzení) V_1, …, V_n soudím, že je pravdivý rovněž výrok V. Zapisujeme schematicky: V_1, …, V_n / V, nebo častěji: [1]



Obrázek 1: Schéma úsudku

Definice: Logické vyplývání

- Úsudek P_1, …, P_n / Z je deduktivně správný (platný), značíme P_1, …, P_n |= Z, jestliže závěr Z logicky vyplývá z předpokladů P_1, …, P_n, tj. za všech okolností takových, že jsou pravdivé všechny předpoklady P_1, …, P_n, je (za těchto okolností) pravdivý i závěr Z.
- Tedy jinými slovy: Za žádných okolností, nikdy se nemůže stát, aby byly všechny předpoklady P_1, …, P_n pravdivé a zároveň závěr Z byl nepravdivý. Závěr Z je pravdivý za všech okolností takových, za kterých jsou pravdivé všechny předpoklady. [1]

2.1.1 Predikátová logika prvního řádu

Predikátová logika prvního řádu (PL1) umožňuje analyzovat elementární výrok až do úrovně vlastností jednotlivých objektů zájmů (tzv. individuí - prvků univerza diskursu) a jejich vztahů. Výrokem rozumíme tvrzení, o němž má smysl prohlásit, zda je pravdivé, či nepravdivé. [1]

V této práci se zabýváme pouze jedním logickým systémem, konkrétně predikátovou logikou prvního řádu, to ale neznamená, že neexistují i jiné logické systémy, např. výroková logika, či predikátové logiky vyšších řádů.

2.1.1.1 Syntaxe predikátové logiky prvního řádu

Syntaxe odpovídá na otázku, jak správně tvořit konstrukce daného jazyka. Abychom mohli vůbec uvažovat formule PL1, musíme znát její syntaxi, kterou popíšeme definicí jazyka PL1.

Definice: Jazyk predikátové logiky prvního řádu

- Abeceda PL1 je tvořena následujícími skupinami symbolů:
  a. Logické symboly:
     i. předmětové (individuové) proměnné: $x, y, z, \ldots$ (případně s indexy)
     ii. symboly pro spojky: $\neg, \wedge, \vee, \supset, \equiv$
     iii. symboly pro kvantifikátory: $\forall, \exists$
     iv. případně binární predikátový symbol = (predikátová logika s rovností)
  b. Speciální symboly (určují specifiku jazyka):
     i. predikátové symboly: P, Q, R, … (případně s indexy)
     ii. funkční symboly: f, g, h, … (případně s indexy)
     Ke každému funkčnímu a predikátovému symbolu je přiřazeno nezáporné číslo n ($n \geq 0$), tzv. arita, udávající počet individuových proměnných, které jsou argumenty funkce nebo predikátu.
  c. Pomocné symboly - závorky: (,),[,],{,}
- Gramatika, která udává, jak správně tvořit:
  a. Termy:
     i. každý symbol proměnné je term
     ii. jsou-li $t_1, \ldots, t_n$ ($n \geq 0$) termy a je-li f n-ární funkční symbol, pak výraz $f(t_1, \ldots, t_n)$ je term; pro $n = 0$ se jedná o nulární funkční symbol, neboli individuovou konstantu (značíme a, b, c, …)
     iii. jen výrazy dle i. a ii. jsou termy
  b. Atomické formule:
     i. je-li p n-ární predikátový symbol a jsou-li $t_1,\ldots,t_n$ termy, pak výraz $p(t_1, \ldots, t_n)$ je atomická formule
     ii. jsou-li t_1 a t_2 termy, pak výraz $(t_1 = t_2)$ je atomická formule
  c. Formule:
     i. každá atomická formule je formule
     ii. je-li výraz A formule, pak $\neg$A je formule
     iii. jsou-li výrazy A a B formule, pak výrazy $(A \vee B)$, $(A \wedge B)$, $(A \supset B)$, $(A \equiv B)$ jsou formule
     iv. je-li x proměnná a A formule, pak výrazy $\forall x$A a $\exists x$A jsou formule
     v. jen výrazy dle i. – iv. jsou formule [1]

Definice: Hierarchický řád formule

- Maximální počet do sebe vnořených závorkových dvojic (,) vyskytujících se ve formuli udává hierarchický řád formule. [1]

Definice: Výskyty proměnných a existenční uzávěr

- Výskyt proměnné x ve formuli A je vázaný, jestliže x je součástí nějaké podformule $\forall x$ B(x) nebo $\exists x$ B(x) formule A.
- Proměnná x je vázaná ve formuli A, má-li v A vázaný výskyt. Výskyt proměnné x ve formuli A, který není vázaný, nazýváme volný.
- Proměnná x je volná ve formuli A, má-li v A volný výskyt.
- Formule se nazývá uzavřenou, neobsahuje-li žádnou volnou proměnnou.
- Nechť $x_1, x_2, \ldots, x_n$ jsou všechny volné proměnné formule A. Potom uzavřenou formuli $\forall A =_{df} \forall x_1 \forall x_2 \ldots \forall x_n A$ resp. $\exists A =_{df} \exists x_1 \exists x_2 \ldots \exists x_n A$, nazýváme generálním resp. existenčním uzávěrem formule A. [1]

2.1.1.2 Sémantika predikátové logiky prvního řádu

Sémantika obecně určuje význam konstrukcí konkrétního jazyka. Sémantika formulí PL1 je dána jejich interpretací, též někdy nazývanou jako interpretační struktura. Jinak řečeno, až konkrétní interpretační struktura dává formuli konkrétní význam.

Definice: Interpretace jazyka PL1

Interpretace jazyka predikátové logiky prvního řádu je tato trojice, která je někdy nazývána interpretační struktura:

1. Neprázdná množina M, která se nazývá universum diskursu a její prvky jsou individua.
2. Interpretace funkčních symbolů jazyka, která přiřazuje každému n-árnímu funkčnímu symbolu f určité zobrazení f_M: $M^n \rightarrow M$.
3. Interpretace predikátových symbolů jazyka, která přiřazuje každému n-árnímu predikátovému symbolu p jistou n-ární relaci p_M nad M, tj. podmnožinu Kartézského součinu M^n. [1]

Definice: Ohodnocení (valuace)

- Ohodnocení individuových proměnných je zobrazení e, které každé proměnné x přiřazuje hodnotu e(x) $\in$ M.
- Ohodnocení termů e* indukované ohodnocením e je induktivně definováno takto:
  - e*(x) = e(x)

- $e^*(f(t_1, t_2, \ldots, t_n)) = f_M(e^*(t_1), e^*(t_2), \ldots, e^*(t_n))$, kde f_M je funkce přiřazená v dané interpretaci funkčnímu symbolu f. [1]

Definice: Pravdivost formule v interpretaci

Pravdivost formule A v interpretaci I pro ohodnocení e individuových proměnných (značíme $\models_I A[e]$ - formule A je pravdivá v I pro e, nebo také A je splněna v I ohodnocením e), je definována v závislosti na tvaru formule:

1. Je-li A atomická formule tvaru
   a) $p(t_1, \ldots, t_n)$, kde p je predikátový symbol (různé od =) a $t_1, \ldots, t_n$ jsou termy, pak $\models_I A[e]$, jestliže platí $\langle e^*(t_1), e^*(t_2), \ldots, e^*(t_n)\rangle \in p_M$, kde p_M je relace přiřazená intepretaci I symbolu p.
   b) $(t_1 = t_2)$, pak $\models_I A[e]$, jestliže platí $e^*(t_1) = e^*(t_2)$, tj. oba termy jsou realizovaný týmž individuem.
2. Je-li A složená formule, tj. je-li tvaru
   c) $\neg B$, pak $\models_I A[e]$, jestliže neplatí $\models_I B[e]$
   d) $B \wedge C$, pak $\models_I [e]$, jestliže platí $\models_I B[e]$ a $\models_I C[e]$
   e) $B \vee C$, pak $\models_I [e]$, jestliže platí $\models_I B[e]$ nebo $\models_I C[e]$
   f) $B \supset C$, pak $\models_I [e]$, jestliže neplatí $\models_I B[e]$ nebo platí $\models_I C[e]$
   g) $B \equiv C$, pak $\models_I [e]$, jestliže platí $\models_I B[e]$ a $\models_I C[e]$, nebo neplatí $\models_I B[e]$ a neplatí $\models_I C[e]$
3. Je-li A formule tvaru
   h) $\forall x$ B, pak $\models_I A[e]$, jestliže pro libovolné individuum $i \in M$ platí $\models_I B[e(x/i)]$, kde $e(x/i)$ je valuace stejná jako e až na to, že přiřazuje proměnné *x* individuum i.
   i) $\exists x$ A, pak $\models_I A[e]$, jestliže pro alespoň jedno individuum $i \in M$ platí $\models_I B[e(x/i)]$, kde $e(x/i)$ je valuace stejná jako e až na to, že přiřazuje proměnné *x* individuum i. [1]

Definice: Splnitelnost formulí PL1

- Formule A je splnitelná, jestliže existuje intepretace I, ve které je splněna, tj. jestliže existuje intepretace I a valuace e takové, že $\models_I A[e]$.
- Formule A je tautologií (logicky pravdivá), značíme $\models A$, jestliže je pravdivá v každé intepretaci.
- Formule A je kontradikce (nesplnitelná), jestliže nemá model, tedy neexistuje intepretace I, která by formuli A splňovala. [1]

2.1.2 Obecná rezoluční metoda

Jednou z metod automatického dokazováni v PL1 je obecná rezoluční metoda, která parciálně rozhoduje, zda je daná formule PL1 kontradikce. Pro předloženou formuli A, která nesplnitelná je, tedy procedura v konečném čase tuto skutečnost zjistí. Pokud je vstupní formule splnitelná, pak algoritmus nemusí svou činnost nikdy ukončit. Chceme-li dokázat, že je daná formule logicky pravdivá, znegujeme ji a zjišťujeme, zda je nesplnitelná. [1]

Rezoluční metoda využívá dvou jednoduchých tvrzení:

1. Je-li formule A tautologie, pak formule $\neg$A je kontradikce a naopak.
2. Rezoluční pravidlo odvozování: Nechť l je literál, potom platí: $(A \vee l) \wedge (B \vee \neg l) \models (A \vee B)$. [1]

2.1.2.1 Skolemova forma a skolemizace

Skolemova forma je významná především v kontextu obecné rezoluční metody, kterou lze aplikovat pouze na formule v klauzulární (Skolemově) formě. Skolemovu formu formule A označujeme A^S. [1]

Definice: Prenexní tvar formule

Formule A predikátové logiky je v prenexním tvaru, má-li podobu $Q_1x_1\ Q_2x_2 \ldots Q_nx_n$ B, kde

- $n \geq 0$ a pro každé i = 1, 2, …, n je Q_i buď všeobecný, nebo existenční kvantifikátor,
- $x_1, x_2, \ldots, x_n$ jsou navzájem různé individuové proměnné,
- B je formule utvořená z elementárních formulí pouze užitím výrokových funktorů $\neg$, $\wedge$, $\vee$.

Výraz $Q_1x_1\ Q_2x_2 \ldots Q_nx_n$ se nazývá prefix a B otevřeným jádrem (maticí). [1]

Definice: Skolemova forma

Skolemova forma uzavřené formule je prenexní tvar této formule, který neobsahuje žádné existenční kvantifikátory. Skolemova forma vznikne z prenexní formy opakovaným použitím následujících dvou operací (skolemizací):

1. $\forall x_1 \ldots \forall x_n\ \exists y\ A(x_1, \ldots, x_n, y) \rightarrow \forall x_1 \ldots \forall x_n\ A(x_1, \ldots, x_n, f(x_1, \ldots, x_n))$, kde f je nový (v jazyce dosud nepoužitý) *n*-ární funkční symbol, tzv. Skolemova funkce.
2. $\exists x\ \forall y_1 \ldots \forall y_n\ A(x, y_1, \ldots, y_n) \rightarrow \forall y_1 \ldots \forall y_n\ A(c, y_1, \ldots, y_n)$, kde c je nová (v jazyce dosud nepoužitá) konstanta, tzv. Skolemova konstanta. [1]

Definice: Klauzulární forma

- Literál je atomická formule nebo negace atomické formule.

- Klausule je disjunkce literálů.
- Klauzulární forma formule je Skolemova forma, jejíž matice je v klauzulárním tvaru, tj. je konjunkcí klausulí. [1]

2.1.2.2 Algoritmus převedení formule do Skolemovy formy

Algoritmus pro převod formule do klauzulární Skolemovy formy se skládá z devíti kroků, přičemž algoritmus jako celek nezachovává pravdivost formule, ale jen její splnitelnost. Pouze splnitelnost zachovávají konkrétně kroky jedna a sedm. Formálně můžeme tuto skutečnost napsat takto: Každá formule A může být převedena na formuli A^S v klauzulární (Skolemově) formě takovou, že A je splnitelná, právě když A^S je splnitelná. Platí tedy vztah $A^S \models A$. [1]

Algoritmus se skládá z těchto devíti kroků: [1]

1. Utvoření existenčního uzávěru formule A.
2. Eliminace nadbytečných kvantifikátorů. Z formule A vypustíme všechny kvantifikátory $\forall x_i$, $\exists x_i$, v jejichž rozsahu se nevyskytuje proměnná x_i.
3. Přejmenování proměnných. Přejmenujeme všechny proměnné, které jsou v A kvantifikovány více než jednou tak, aby všechny kvantifikátory měly navzájem různé proměnné.
4. Eliminace spojek $\supset, \equiv$ podle těchto vztahů:
   $(A \supset B) \Leftrightarrow (\neg A \vee B)$, $(A \equiv B) \Leftrightarrow (\neg A \vee B) \wedge (\neg B \vee A)$.
5. Přesun spojek $\neg$ dovnitř.
6. Přesun kvantifikátorů doprava. Provádíme náhrady podle těchto ekvivalencí (Q je kvantifikátor $\forall$ nebo $\exists$, © je symbol $\wedge$ nebo $\vee$; A, B neobsahují volnou proměnnou x):
   $Qx\ (A \copyright B(x)) \Leftrightarrow A \copyright Qx\ B(x)$, $Qx\ (A(x) \copyright B) \Leftrightarrow Qx\ A(x) \copyright B$
7. Eliminace existenčních kvantifikátorů, tzv. Skolemizace. Provádíme postupně Skolemizaci podformulí $Qx\ B(x)$, $Qx\ A(x)$, které jsme obdrželi v předchozím kroku 6, tedy náhradu existenčně kvantifikovaných formulí formulemi bez existenčního kvantifikátoru podle definice, která byla uvedena výše,
8. Přesun všeobecných kvantifikátorů doleva.
9. Použití distributivních zákonů pro převedení do formule do klauzulární formy.
   Provedeme postupně náhrady vlevo formulemi vpravo:
   $(A \wedge B) \vee C \Leftrightarrow (A \vee C) \wedge (B \vee C)$, $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

Krok šest je důležitý z důvodu, že minimalizuje počet argumentů v následně zaváděných Skolemových funkcích (konstantách). Vynechání tohoto kroku může tedy vést ke zbytečně velkému počtu argumentů Skolemových funkcí, což může negativně ovlivnit následnou rezoluční metodu (unifikační algoritmus) tak, že dostaneme chybný výsledek. Nebudeme pak schopni dokázat nesplnitelnost negované vstupní formule, a tudíž ani nedokážeme tautologičnost samotné vstupní formule. [1]

Důležitost kroku 6 demonstrujeme na následujících dvou jednoduchých příkladech:

1. Uvažujme formuli $A = \forall x\ \exists y\ \forall z\ \exists v\ [P(z, y) \wedge Q(x, v)]$. Pokud bychom aplikovali Skolemizaci bez kroku 6, dostali bychom formuli: $A^{S'} = \forall x\ \forall z\ [P(z, f(x) \wedge Q(x, h(x, z)]$, kde f, h jsou zavedené Skolemovy funkce. Použijeme-li však nejprve krok 6, dostaneme:
   $A' = \exists y\ \forall z\ P(z, y) \wedge \forall x\ \exists v\ Q(x, v)$ a z ní eliminací existenčních kvantifikátorů $A'' = \forall z\ P(z, a) \wedge \forall x\ Q(x, h(x))$. Odtud pak přesunem kvantifikátorů doleva: $A^S = \forall z\ \forall x\ [P(z, a) \wedge Q(x, h(x))]$, v níž zavedené Skolemovy funkce a, h jsou jednodušší. [1]
2. Uvažujme jednoduchou tautologii $\neg\forall x\ P(x) \vee \forall y\ P(y)$. Negací získáme formuli $\forall x\ P(x) \wedge \exists y\ \neg P(y)$, která je nesplnitelná. Její nesplnitelnost snadno dokážeme. Skolemizací obdržíme formuli $\forall x\ P(x) \wedge \neg P(a)$ a substitucí $\{x / a\}$ pak kontradikci $P(a) \wedge \neg P(a)$, tedy prázdnou klausuli. Provedeme-li chybně nejprve převod do prenexní formy, a pak Skolemizaci, dostaneme $\forall x\ P(x) \wedge \exists y\ \neg P(y)] \vdash \forall x\ [P(x) \wedge \exists y\ \neg P(y)] \vdash \forall x \exists y\ [P(x) \wedge \neg P(y)] \vdash \forall x\ [P(x) \wedge \neg P(f(x))$. Nesplnitelnost této formule nedokážeme, protože literály $P(x)$ a $\neg P(f(x))$ nejsou unifikovatelné. [1]

2.1.2.3 Unifikace

Uplatnění rezolučního pravidla je v PL1 ztíženo tím, že se se v jednotlivých literálech vyskytují termy obecně různého tvaru, které je nutno unifikovat.

Definice: Substituce, unifikace a nejobecnější unifikace

- Nechť A je formule obsahující individuové proměnné x_i, i = 1, 2, ..., n. Označme $\sigma = \{x_1/t_1, x_2/t_2, ..., x_n/t_n\}$ simultánní substitucí termů t_i předmětové proměnné x_i pro i = 1, 2, ..., n. Potom zápisem $A\sigma$ označíme formuli, která vznikne z formule A provedením substituce σ.
- Unifikace formulí A, B je substituce σ taková, že $A\sigma = B\sigma$.
- Nejobecnější unifikace formulí A, B je unifikace σ taková, že pro každou jinou unifikaci ρ formulí A, B platí $\rho = \sigma\tau$, kde $\tau \neq \varepsilon$, tj. každá unifikace vznikne z nejobecnější unifikace provedením další dodatečné substituce. [1]

Unifikace literálů A, B nemusí vůbec existovat. V takovém případě jsou literály A, B neunifikovatelné a nemůžeme na ně uplatnit rezoluční pravidlo. [1]

2.2 Formální teorie jazyků

Definice:

- Abeceda je libovolná konečná množina, často označovaná Σ. Prvky abecedy jsou symboly. [2]
- Slovo, též řetězec, nad abecedou Σ je konečná posloupnost prvků Σ. [2]
- Výrazem Σ^* značíme množinu všech nad abecedou Σ. [2]
- Jazyk nad abecedou Σ, je libovolná množina slov v abecedě Σ, tedy libovolná podmnožinou Σ^*. [2]

Další definice se týká pojmu bezkontextová gramatika, kterou můžeme popisovat nebo rozpoznávat jazyky z třídy bezkontextových jazyků.

Definice: Bezkontextová gramatika

Bezkontextová gramatika je uspořádaná čtveřice $G = (\Pi, \Sigma, S, P)$, kde

1. Π je konečná množina neterminálních symbolů, též neterminálů.
2. Σ je konečná množina terminálních symbolů, též terminálů, přičemž $\Pi \cap \Sigma = \emptyset$.
3. $S \in \Pi$ je počáteční (startovací) neterminál.
4. P je konečná množina pravidel typu $A \rightarrow \beta$, kde A je neterminál, tedy $A \in \Pi$ a β je řetězec složený z terminálů a neterminálů, tedy $\beta \in (\Pi \cup \Sigma)^*$. [2]

Definice:

- Jazyk generovaný gramatikou G označujeme L(G). [2]
- Mějme gramatiku $G = (\Pi, \Sigma, S, P)$ a uvažujme řetězec $\gamma, \delta \in (\Pi \cup \Sigma)^*$. Řetězec γ lze přímo přepsat na δ ($\gamma \Rightarrow \delta$), pokud existují slova u_1, u_2 a pravidlo $A \rightarrow \beta$ tak, že $\gamma = u_1 A u_2$ a $\delta = u_1 \beta u_2$. Řetězec γ lze přepsat na δ ($\gamma \Rightarrow^* \delta$), pokud existuje posloupnost $u_0, u_1, \ldots, u_n$ slov z $(\Pi \cup \Sigma)^*$ taková, že $\gamma = u_0 \Rightarrow u_1 \Rightarrow \ldots \Rightarrow u_n = \delta$. Tato posloupnost se pak nazývá odvození (derivací) délky *n* slova δ ze slova γ. Řetězec γ lze přepsat na δ levým přepsáním, jestliže $\gamma = uX\beta$, $\delta = u\alpha\beta$, pro $u \in \Sigma^*$, $\beta \in (\Pi \cup \Sigma)^*$ a pravidlo $X \rightarrow \alpha$ v P. Pokud se odvození skládá pouze z posloupností levých přepsaní, pak jej nazýváme levým odvozením, neboli levou derivací. [2]
- Derivační strom vztahující se k bezkontextové gramatice $G = (\Pi, \Sigma, S, P)$, je uspořádaný kořenový strom, jehož vrcholy jsou ohodnoceny symboly z $\Pi \cup \Sigma \cup \{\varepsilon\}$ a následníci každého vrcholu jsou uspořádáni zleva doprava. Kořen je ohodnocen symbolem S a dále platí, že každý vrchol ohodnocený neterminálem $X \in \Pi$ musí mít následníky, jejichž ohodnocení odpovídá pravé straně nějakého pravidla $X \rightarrow Y_1 Y_2 \ldots Y_n$. Listové vrcholy jsou pak ohodnoceny pouze symboly z $\Sigma \cup \{\varepsilon\}$. Řekneme, že se jedná o derivační strom pro slovo *w*, jestliže *w* je zřetězením ohodnocení listů zleva doprava. [2]

- Bezkontextová gramatika G je jednoznačná, jestliže každé slovo z L(G) má právě jedno levé odvození (tj. právě jeden derivační strom). [2]

Jednoznačnost bezkontextové gramatiky je důležitá zejména při rozpoznávání jazyka, kdy existuje právě jeden derivační strom popisující odvození vstupního slova. Správně navržená bezkontextová gramatika může zachycovat i prioritu vyhodnocování vstupního slova, například prioritu matematických operací nebo logických spojek. S pojmem derivační strom pak úzce souvisí syntaktický strom, který lze na základě znalosti stromu derivačního vytvořit.

Další vlastností bezkontextových gramatik, která najde uplatnění při rozpoznávání vstupního slova *w* je, zda na základě informace, jaký je aktuálně čtený symbol a jaké symboly jsme již přečetli, můžeme jednoznačně konstruovat derivační strom pro dané vstupní slovo *w*. Takovouto vlastnost mají například tzv. LL(1) gramatiky. První L v názvu znamená, že vstupní slovo čteme zleva doprava, druhé L znamená, že konstruujeme levou derivaci a hodnota „1“ vyjadřuje počet symbolů vstupního textu, které musíme znát při rozhodování o průběhu konstrukce derivačního stromu. [3] Pro jejich definici však musíme nejprve ještě definovat množiny FIRST a FOLLOW.

Definice: Množina FIRST

Je-li α řetězec symbolů bezkontextové gramatiky G, potom FIRST(α) je množina terminálních symbolů, jimiž mohou začínat řetězce derivované z α. Pokud $\alpha \Rightarrow^* \varepsilon$, pak rovněž $\varepsilon \in$ FIRST(α). [3]

Definice: Množina FOLLOW

FOLLOW(A) pro neterminál A je množina všech terminálů *a*, které se mohou vyskytovat bezprostředně vpravo od A v nějaké větné formě, tj. množina takových neterminálů, pro něž existuje derivace ve tvaru $S \alpha \Rightarrow^* \alpha A a \beta$ pro nějaké α a β. Může-li být A nejpravějším symbolem v nějaké větné formě, je ve FOLLOW(A) rovněž symbol $, který reprezentuje konec vstupního řetězce. [3]

Definice: LL(1) gramatika

Bezkontextová gramatika je LL(1), jestliže každá množina pravidel se stejnou levou stranou $A \rightarrow \alpha_1 \mid \alpha_2 \ldots \mid \alpha_n$ splňuje následující dvě vlastnosti:

1. FIRST(α_i) $\cap$ FIRST(α_j) = $\emptyset$ pro všechna $i \neq j$, tzv. vlastnost FF
2. Pokud některé $\alpha_i \Rightarrow^* \varepsilon$, pak pro všechna $j \neq i$ FIRST(α_j) $\cap$ FOLLOW(A) = $\emptyset$, tzv. vlastnost FFL

Jednou z metod syntaktické analýzy jazyka je tzv. analýza shora dolů. Tu můžeme charakterizovat jako proces hledání levé derivace vstupního slova, nebo také jako vytváření derivačního stromu počínaje jeho kořenem. Jednou z možností implementace syntaktické analýzy shora dolů je analýza rekurzivním sestupem. Tato metoda spočívá v implementaci

samostatných podprogramů pro analýzu každého neterminálu dané gramatiky. Syntaktická analýza se pak spustí voláním podprogramu odpovídajícímu startovnímu neterminálu bezkontextové gramatiky. [3]

3 Zachycení složitosti úloh pro skolemizaci

V této kapitole se budu zabývat analýzou a návrhem automatizovaného postupu zachycení složitosti úloh pro převedení formule do Skolemovy klauzulární formy, dále označováno jen skolemizací. Skolemizace patří k obtížnější látce probírané v kurzu Matematická logika a je velice žádoucí, aby pedagog mohl dopředu exaktně zjistit, jak je daná úloha náročná na vyřešení, a které jednotlivé kroky algoritmu skolemizace je potřeba použít. Tato informace je důležitá jednak při procvičování skolemizace, protože není vhodné na úvod nechat studenty řešit komplexní příklady, kde musí použít všechna pravidla nebo je možné rychle vybírat příklady na procvičení konkrétního zvoleného kroku, stejně tak při tvorbě zápočtových a zkouškových písemek, kdy je možné vytvořit jednoduše více variant se shodnou obtížností. Různé formule se však vzájemně liší nejen složitostí spočívající v aplikaci konkrétních kroků algoritmu skolemizace, ale složitost úlohy určuje rovněž syntaktická struktura formule, zejména její hierarchický řád.

3.1 Algoritmus zachycení složitosti úlohy

Algoritmus pro zachycení složitosti vychází z [4].

Vstupem algoritmu bude formule PL1 a jeho výstupem bude binární řetězec složitosti. Složky řetězce složitosti budou odpovídat jednotlivým pravidlům používaným při skolemizaci a hierarchickému řádu formule. Bude-li konkrétní pravidlo během skolemizace alespoň jednou použito, potom bude na odpovídající složce v řetězci hodnota „1“. V případě, že pravidlo použito ani jednou nebude, bude daná pozice nabývat hodnoty „0“. Obdobně bude zachycen hierarchický řád formule, pouze s tím rozdílem, že na jeho položce bude nezáporné číslo, odpovídající hodnotě hierarchického řádu.

Pravidla algoritmu skolemizace budou členěna podrobněji než do základních devíti kroků. Důvodem pro detailnější členění je požadavek, aby byla podrobněji zachycena složitost úlohy. Rovněž budeme, jak už bylo výše řečeno, při stanovení složitosti zohledňovat hierarchický řád formule.

Tabulka 1: Složky řetězce složitosti

Číslo složky (pozice)	Popis složky	Dále členěná	Počet složek v řetězci
1	Zavedení existenčního uzávěru	ne	1
2	Eliminace nadbytečných kvantifikátorů	ne	1
3	Přejmenování proměnných	ne	1
4	Eliminace spojek	ano	4
4.1	Eliminace implikace	ne	1
4.2	Eliminace ekvivalence	ne	1
4.3	Eliminace NAND, nebo NOR (volitelně)	ne	1
4.4	Eliminace jiné spojky (volitelně)	ne	1
5	Přesunutí negace doprava	ano	4
5.1	Eliminace vzorů $\neg(A \wedge B)$ nebo $\neg(A \vee B)$	ano	3
5.1.1	Právě jeden z výrazů A, B je literál	ne	1
5.1.2	Výrazy A, B nejsou literály	ne	1
5.2	Přesun negace přes kvantifikátory	ne	1
6	Přesunutí kvantifikátorů doprava	ano	2
6.1	Lze přesunout minimálně jeden, případně všechny	ne	1
6.2	Lze přesunout minimálně jeden, avšak ne všechny	ne	1
7	Skolemizace - eliminace existenčních kvantifikátorů	ano	2
7.1	Skolemizace konstantou	ne	1
7.2	Skolemizace funkcí	ne	1
8	Přesun kvantifikátorů doleva	ne	1
9	Distributivní zákony (převedení na klauzulárni tvar)	ano	5
9.1	A, B jsou literály a C je disjunkce dvou literálů - $(A \wedge B) \vee (D \vee E)$	ne	1
9.2	C je literál, jeden z dvojice A, B je literál a druhý je konjunkcí dvou literálů - $(A \wedge D \wedge E) \vee C$	ne	1
9.3	A, B jsou literály a C je konjunkce dvou literálů - $(A \wedge B) \vee (D \wedge E)$	ne	1
9.4	Alespoň jeden z výrazů A, B, C je složený a nelze jej popsat kroky 9.1 až 9.3.	ne	1
10	Hierarchický řád vstupní formule	ne	1
11	Hierarchický řád formule po převedení do Skolemovy klauzulární formy.	ne	1

Řetězec složitosti bude tedy obsahovat celkem 23 složek.

1	2	3	4.1	4.2	4.3	4.4	5.1	5.1.1	5.1.2	5.2	6.1	6.2	7.1	7.2	8	9	9.1	9.2	9.3	9.4	10	11

3.1.1 Eliminace jiné spojky

Bude možné definovat vlastní logickou spojku odlišnou od spojek negace, konjunkce, disjunkce, implikace, ekvivalence, NAND a NOR. Pro tuto spojku bude nutné rovněž definovat pravidlo pro její eliminaci.

3.1.2 Eliminace výrazů $\neg(A \wedge B)$ a $\neg(A \vee B)$

Při eliminaci výrazů $\neg(A \wedge B)$, $\neg(A \vee B)$ pomocí De-Morganových zákonů budeme z hlediska složitosti rozlišovat tyto tři situace:

1. Výrazy A, B jsou literály. Za této situace se jedná o prostou a jednoduchou aplikaci dle příslušných základních vzorců:
   - $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$
   - $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$

   Příklad:

   $$\neg(P(x) \wedge Q(y)) \Leftrightarrow \neg P(x) \vee \neg Q(y)$$

   Jedná se o eliminaci výrazu $\neg(A \wedge B)$, kdy A, B jsou literály, tedy aplikace De Morganových zákonů na nejjednodušší možný případ, v řetězci složitosti se projeví hodnotou „1“ pouze na pozici 5.1.

2. Právě jeden z výrazů A, B je literálem. Jelikož pro konjunkci i disjunkci platí komutativní zákon, není podstatné, který z dvojice výrazů A, B je literál. Tento případ zachycuje druhou nejjednodušší možnou situaci při eliminaci výrazů $\neg(A \wedge B)$, $\neg(A \vee B)$, ale již takovou, kde není možné bez hlubšího pochopení problému jen de-facto opsat základní vzorce uvedené u případu prvního.

   Pokud se v řetězci složitosti objeví hodnota „1“ na pozici 5.1.1, pak bude hodnota „1“ automaticky i na pozici 5.1, jelikož krok 5.1.1 je speciálním případem kroku 5.1.

   Příklad:

   $$\neg[(P(x) \wedge Q(y)) \vee R(z)] \Leftrightarrow$$
   $$\neg(P(x) \wedge Q(y)) \wedge \neg R(z) \Leftrightarrow (\neg P(x) \vee \neg Q(y)) \wedge \neg R(z)$$

   Výraz $R(z)$ je literálem a výraz $(P(x) \wedge Q(y))$ literálem není. Jedná se tedy o případ druhý a v řetězci složitosti se projeví hodnotou „1“ na pozicích 5.1.1 a 5.1.

   Příklad:

   $$\neg[\neg(P(x) \wedge Q(y)) \vee R(z)] \Leftrightarrow$$
   $$\neg\neg(P(x) \wedge Q(y)) \wedge \neg R(z) \Leftrightarrow (P(x) \wedge Q(y)) \wedge \neg R(z)$$

Výraz R(z) je literálem a výraz ($\neg$(P(x) $\wedge$ Q(y))) je negace konjunkce dvou literálů, čili není literál. Jedná se tedy o případ druhý a v řetězci složitosti se projeví hodnotou „1“ na pozici 5.1.1.

3. Poslední a pro studenty nejobtížnější případ bude takový, ve kterém oba výrazy A, B nejsou literály.

Podobně jako v předchozím případě bude platit, že pokud se v řetězci složitosti objeví hodnota „1“ na pozici 5.1.2, pak bude hodnota „1“ automaticky i na pozici 5.1, jelikož krok 5.1.2 je opět speciálním případem kroku 5.1.

Příklad:

$$\neg\{[P(x) \wedge Q(y)] \vee [R(z) \vee S(u)]\} \Leftrightarrow$$
$$\neg[P(x) \wedge Q(y)] \wedge \neg[R(z) \vee S(u)] \Leftrightarrow$$
$$[\neg P(x) \vee \neg Q(y)] \wedge [\neg R(z) \wedge \neg S(u)]$$

Výrazy [P(x) $\wedge$ Q(y)] a [R(z) $\vee$ S(u)] nejsou literály, jedná se tedy o případ třetí a v řetězci složitosti se projeví na vrcholu 5.1.2 a 5.1.

Zavádět více případů by již příliš zkomplikovalo pochopení jejich rozlišení a nepřineslo by téměř žádnou přidanou hodnotu. Poslední obecný případ totiž již ve své nejjednodušší instanci dostatečně prověřuje, zda student do hloubky rozumí probírané látce.

3.1.3 Členění přesunů kvantifikátorů doprava

Krok šestý (přesun kvantifikátorů doprava) budeme dělit na tyto dva případy:

1. Lze přesunout minimálně jeden, případně všechny.

Příklad:

$$\forall x\,[P(x) \wedge \exists y\,Q(y)] \Leftrightarrow \forall x\,P(x) \wedge \exists y\,Q(y)$$

Přesun všeobecného kvantifikátoru vázajícího proměnnou x je možný. Tato úprava se projeví v řetězci složitosti hodnotou „1“ na pozici 6.1.

2. Lze přesunout minimálně jeden, avšak ne všechny.

Příklad:

$$\forall x\,\exists y\,[P(x, y) \wedge Q(y)]$$

Existenční kvantifikátor vázající proměnnou y nelze přes konjunkci přesunout. Tato situace se projeví v řetězci složitosti hodnotou „1“ na pozici 6.2.

3.1.4 Eliminace existenčních kvantifikátorů - Skolemizace

Při dělení kroku sedmého budeme rozlišovat, zda provádíme skolemizaci funkcí, nebo konstantou.

1. Skolemizace konstantou

   Příklad:

   $$\exists x\ \mathrm{Q}(x) \vdash \mathrm{Q}(a)$$

   Jedná se o Skolemizaci konstantou. Tato úprava se projeví v řetězci složitosti hodnotou „1“ na pozici 7.1.

2. Skolemizace funkcí

   Příklad:

   $$\forall x\ \exists y\ \mathrm{Q}(x, y) \vdash \forall x\ \mathrm{Q}(x, \mathrm{f}(x))$$

   Jedná se o Skolemizaci funkcí. Tato úprava se projeví v řetězci složitosti hodnotou „1“ na pozici 7.2.

3.1.5 Rozlišení aplikací distributivních zákonů

U kroku osmého, tedy aplikace distributivních zákonů, budeme používat úpravy podle vzorce $(\mathrm{A} \wedge \mathrm{B}) \vee \mathrm{C} \Leftrightarrow (\mathrm{A} \vee \mathrm{C}) \wedge (\mathrm{B} \vee \mathrm{C})$ a z hlediska složitosti rozlišovat těchto pět případů:

1. První, nejjednodušší případ bude prostou aplikací vzorce $(\mathrm{A} \wedge \mathrm{B}) \vee \mathrm{C} \Leftrightarrow (\mathrm{A} \vee \mathrm{C}) \wedge (\mathrm{B} \vee \mathrm{C})$, všechny tří výrazy A, B a C jsou tedy literály.

   Příklad:

   $$(\mathrm{P}(x) \wedge \mathrm{Q}(y)) \vee \mathrm{R}(z) \Leftrightarrow (\mathrm{P}(x) \vee \mathrm{R}(z)) \wedge (\mathrm{Q}(y) \vee \mathrm{R}(z))$$

   V tomto případě byly všechny tři výrazy literály, Tato situace se projeví hodnotou „1“ v řetězci složitosti pouze na pozici 9.

2. Druhým případem bude situace, kde výraz C je disjunkcí právě dvou literálů, které označíme D a E. Celý jej pak můžeme popsat vzorem $(\mathrm{A} \wedge \mathrm{B}) \vee (\mathrm{D} \vee \mathrm{E})$.

   Příklad:

   $$(\mathrm{P}(x) \wedge \mathrm{Q}(y)) \vee (\mathrm{R}(z) \vee \mathrm{Q}(u)) \Leftrightarrow$$
   $$[(\mathrm{P}(x) \vee (\mathrm{R}(z) \vee \mathrm{Q}(u))] \wedge [\mathrm{Q}(y) \vee (\mathrm{R}(z) \vee \mathrm{Q}(u))]$$

Výraz $P(x) \wedge Q(y)$ je konjunkce dvou literálů a výraz $R(z) \vee Q(u)$ je disjunkce dvou literálů, jedná se tedy o případ druhý a v řetězci složitosti se projeví hodnotou „1“ na pozicích 9.1 a 9.

Kroky 9.1 až 9.4 jsou speciálními případy kroku 9 (distributivní zákony). Proto, bude-li hodnota „1“ na kterékoliv pozici 9.1 až 9.5, bude automaticky hodnota „1“ i na pozici 9.

3. Ve třetím případě bude právě jeden z dvojice výrazů A, B konjunkcí právě dvou literálů, které označíme D, E a případ můžeme popsat vzorem $(A \wedge D \wedge E) \vee C$. Druhý a třetí případ jsou si podobně a jedná se de-facto o druhou nejjednodušší možnost, která může při aplikaci distributivních zákonů nastat.

Příklad:

$$[(P(x) \wedge Q(y)) \wedge R(z)] \vee Q(u) \Leftrightarrow$$
$$(P(x) \vee Q(u)) \wedge [(Q(y) \vee Q(u)) \wedge (R(z) \vee Q(u))]$$

Výraz $(P(x) \wedge Q(y)) \wedge (R(z)$ je konjunkcí tří literálů, který je v disjunkci s právě jedním literálem. Jedná se o případ třetí, který se v řetězci složitosti projeví na pozici 9.2 a 9.

4. Ve čtvrtém případě budou výrazy A, B literály a C konjunkce právě dvou literálů, zapsáno pomocí vzoru $(A \wedge B) \vee (D \wedge E)$, kde D a E jsou literály. Tato situace je již na vyřešení obtížnější.

Příklad:

$$(P(x) \wedge Q(y)) \vee (R(z) \wedge Q(u)) \Leftrightarrow$$
$$[(P(x) \wedge Q(y)) \vee R(z)] \wedge [(P(x) \wedge Q(y)) \vee Q(u)] \Leftrightarrow$$
$$[(P(x) \vee R(z)) \wedge (Q(y)) \vee R(z))] \wedge [(P(x) \vee Q(u)) \wedge (Q(y)) \vee Q(u))]$$

Tento příklad zachycuje disjunkci dvou konjunkcí dvou literálů, popsanou jako případ čtvrtý. Příklad se v řetězci složitosti projeví hodnotou „1“ na pozici 9.3 a 9.

5. Poslední případ bude postihovat zcela obecnou situaci, kdy alespoň jeden z výrazů A, B, C je složený a nelze jej popsat výše uvedenými. Tento případ klade největší nároky na znalosti studenta.

Příklad:

$$[(P(x) \wedge Q(y)) \vee P(u)] \vee R(z) \Leftrightarrow [(P(x) \vee P(u)) \wedge (Q(y)) \vee P(u))] \vee R(z)$$
$$\Leftrightarrow [(P(x) \vee P(u)) \vee R(z))] \wedge [(Q(y)) \vee P(u)) \vee R(z)]$$

V tomto příkladu se jedná o případ pátý. Příklad se v řetězci složitosti projeví hodnotou „1“ na pozici 9.4 a 9.

3.1.6 Klasifikace dle hierarchického řádu formule

V rámci algoritmu zachycení složitosti úlohy budeme pracovat se dvěma hodnotami hierarchického řádu formule. První je hodnota hierarchického řádu vstupní formule, to je formule, kterou chceme převést do Skolemovy klauzulární formy. Druhou potom hodnota hierarchického řádu výsledné formule, to je takové, která vznikla převedením do Skolemovy klauzulární formy.

Příklad:

$$\exists x \, \mathrm{P}(x)$$

Převedením do Skolemovy klauzulární formy dostaneme opět formuli $\exists x \, \mathrm{P}(x)$, jejíž hierarchický řád je nula. Na pozici 10 a 11 řetězce složitosti budou tedy hodnoty „0“.

Příklad:

$$\neg\exists x \, (\neg\mathrm{P}(x) \vee \neg\mathrm{Q}(x))$$

Hierarchický řád příkladu je tři. Převedením do Skolemovy klauzulární formy dostaneme formuli $\forall x \, (\mathrm{P}(x) \wedge \mathrm{Q}(x))$, jejíž hierarchický řád je jedna. Na pozici 10 v řetězci složitosti bude hodnota „3“ a na pozici 11 hodnota „1“.

3.2 Návrh systému pro zachycení složitosti

Systém dostane na vstupu formuli v PL1 a na výstupu vrátí řetězec složitosti skolemizace. Systém se skládá ze dvou hlavních částí. První subsystém má za úkol lexikální a syntaktickou analýzu zadané formule a jeho výstupem je její syntaktický strom, který je zároveň vstupem pro druhý subsystém, jehož úkolem je na vstupní syntaktický strom postupně uplatňovat jednotlivá pravidla skolemizace a zároveň si pamatovat, která pravidla použil. Systém tak bude automatizovaně provádět skolemizaci Nakonec pak sestaví řetězec složitosti skolemizace.

3.2.1 Syntaktický analyzátor a gramatika

Činnost syntaktického analyzátoru vstupních formulí je založena na analýze shora dolů, konkrétně na principu analýzy rekurzivním sestupem. Pro potřebu samotné implementace analýzy rekurzivním sestupem je však potřeba nejprve navrhnout jednoznačnou gramatiku, která bude schopna rozpoznávat všechny dobře utvořené formule z vymezené podmnožiny jazyka PL1. Tato gramatika by rovněž měla být LL(k), kde k by mělo být co nejmenší, v optimálním případě rovno „1“, aby byl syntaktický analyzátor co možná nejjednodušší.

Gramatiku rozpoznávající dobře utvořené formule navrhneme následovně:

1. Množina neterminálních symbolů $\Pi = \{S, T, E, K, P, I, A, A', J, F, X, B, C, D\}$.

2. Množina terminálních symbolů $\Sigma = \{a, c, f, x, ,, (,), \forall, \exists, \supset, \equiv, \wedge, \vee, \downarrow, \uparrow\}$, kde x je identifikátor proměnné, c je identifikátor konstanty, f je identifikátor funkce a a je identifikátor predikátu.
3. Počátečním neterminálem bude neterminál S.
4. Množina pravidel P gramatiky bude následující.

S	$\rightarrow$	ET	
T	$\rightarrow$	$\supset E \mid \equiv E \mid \wedge E \mid \vee E \mid \downarrow E \mid \uparrow E \mid \varepsilon$	
E	$\rightarrow$	$KE \mid P \mid (S) \mid \neg E$	
K	$\rightarrow$	$\forall X \mid \exists X$	
P	$\rightarrow$	BI	
I	$\rightarrow$	$(A \mid \varepsilon$	
A	$\rightarrow$	JA'	
A'	$\rightarrow$	$,JA' \mid)$	
J	$\rightarrow$	$F \mid C \mid X$	
F	$\rightarrow$	$D(A$	
X	$\rightarrow$	x	– identifikátor proměnné
B	$\rightarrow$	a	– identifikátor predikátu
C	$\rightarrow$	c	– identifikátor konstanty
D	$\rightarrow$	f	– identifikátor funkce

3.2.1.1 LL(1) vlastnost navržené gramatiky, množiny first a follow

1. $S \rightarrow ET$
   First(ET) = { $\neg, \exists, \forall, (, a$ }
   Follow(S) = { $\$,)$ }

2. $T \rightarrow \supset E \mid \equiv E \mid \wedge E \mid \vee E \mid \downarrow E \mid \uparrow E \mid \varepsilon$
   First($\supset E$) = { $\supset$ }
   First($\equiv E$) = { $\equiv$ }
   First($\wedge E$) = { $\wedge$ }
   First($\vee E$) = { $\vee$ }
   First($\downarrow E$) = { $\downarrow$ }
   First($\uparrow E$) = { $\uparrow$ }
   First(ε) = { ε }
   Follow(T) = { $\$,)$ }
   First($\supset E$) $\cap$ First($\equiv E$) $\cap$ First($\wedge E$) $\cap$ First($\vee E$) $\cap$ First($\downarrow E$) $\cap$ First($\uparrow E$) $\cap$ Follow(T) = $\emptyset$

3. $E \rightarrow KE \mid P \mid (S) \mid \neg E$

   First(KE) = { $\exists$, $\forall$ }
   First($\neg E$) = { $\neg$ }
   First(P) = { a }
   First((S)) = { (}
   Follow(E) = { $,), $\supset$, $\equiv$, $\wedge$, $\vee$, $\downarrow$, $\uparrow$ }
   First(KE) $\cap$ First(P) $\cap$ First((S)) $\cap$ First($\neg E$) = $\emptyset$

4. $K \rightarrow \forall X \mid \exists X$

   First($\forall X$) = { $\forall$ }
   First($\exists X$) = { $\exists$ }
   Follow(K) = { a, (,$\exists$, $\forall$, $\neg$ }
   First($\forall X$) $\cap$ First($\exists X$) = $\emptyset$

5. $P \rightarrow BI$

   First(BI) = { a }
   Follow(P) = { $,), $\supset$, $\equiv$, $\wedge$, $\vee$, $\downarrow$, $\uparrow$ }

6. $I \rightarrow (A \mid \varepsilon)$

   First($(A$) = { (}
   First(ε) = { ε }
   Follow(I) = { $,), $\supset$, $\equiv$, $\wedge$, $\vee$, $\downarrow$, $\uparrow$ }
   First($(A$) $\cap$ First(ε) $\cap$ Follow(I) = $\emptyset$

7. $A \rightarrow JA'$

   First(JA') = {f, c, x }
   Follow(A) = { , , $,), $\supset$, $\equiv$, $\wedge$, $\vee$, $\downarrow$, $\uparrow$ }

8. $A' \rightarrow ,JA' \mid)$

   First(,JA') = { , }
   First()) = {) }
   Follow(A') = { , , $,), $\supset$, $\equiv$, $\wedge$, $\vee$, $\downarrow$, $\uparrow$ }
   First(,JA') $\cap$ First()) = $\emptyset$

9. $J \rightarrow F \mid C \mid X$

   First(F) = {f }
   First(C) = { c }
   First(X) = { x }
   Follow(J) = { , ,) }
   First(F) $\cap$ First(C) $\cap$ First(X) = $\emptyset$

10. $F \rightarrow D(A$

    First($D(A$) = {f }
    Follow(F) = { , ,) }

11. $X \rightarrow x$

    First(X) = { x }
    Follow(X) = { , (,), $\neg$, $\exists$, $\forall$, a }

12. $B \rightarrow a$
    First(B) = { a }
    Follow(B) = { $, (,), \supset, \equiv, \wedge, \vee, \downarrow, \uparrow$ }

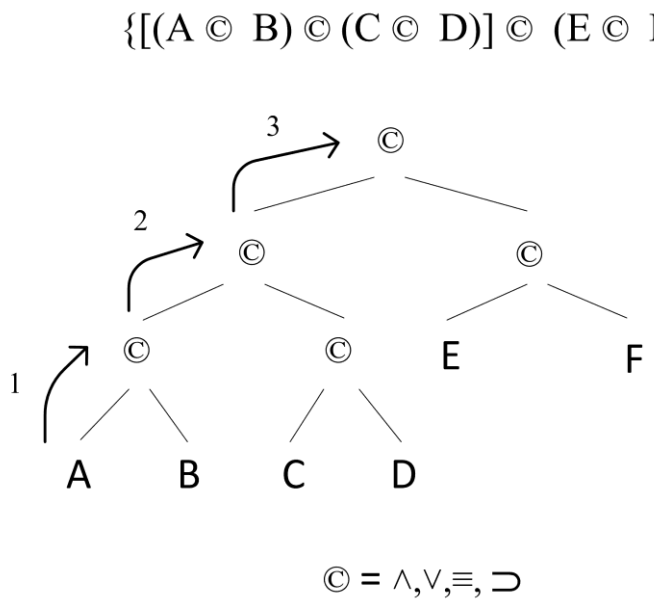
13. $C \rightarrow c$
    First(C) = { c }
    Follow(C) = { , ,) }

14. $D \rightarrow f$
    First(D) = { f }
    Follow(D) = { (}

Gramatika je LL(1), jelikož u žádného pravidla nedochází k porušení vlastností FF a FFL, které jsme definovali v kapitole, věnované formální teorii jazyků. Identifikátory predikátu, funkcí, konstant a proměnných (a, f, c, x) bude možné parametrizovat formou regulárních výrazů.

3.2.2 Výpočet hierarchického řádu formule

Máme-li k dispozici syntaktický strom formule, její hierarchický řád vypočítáme následovně. Pro každý vrchol p syntaktického stromu reprezentující predikát zjistíme počet vrcholů na cestě od vrcholu p ke kořeni takových, že reprezentují logickou spojku. Maximum z těchto hodnot potom odpovídá hierarchickému řádu formule.



Obrázek 2: Příklad výpočtu hierarchického řádu ze syntaktického stromu

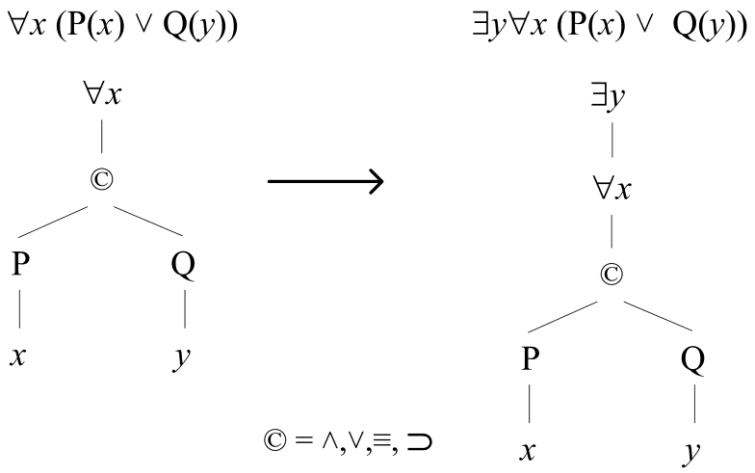
V příkladu na obrázku nalezneme na cestě od vrcholu predikátu ke kořenu maximálně tři logické spojky, čemuž je roven hierarchický řád formule.

3.2.3 Aplikace kroků skolemizace na syntaktický strom

Pro jednotlivé kroky skolemizace je potřeba přesně specifikovat algoritmy, které příslušně modifikují syntaktický strom vstupní formule. V rámci některých kroků skolemizace chceme jemněji rozlišit, které pravidlo bylo použito. Proto musíme při analýze těchto kroků do detailu rozebrat každou možnou situaci.

3.2.3.1 Utvoření existenčního uzávěru formule

Procházíme syntaktickým stromem do šířky, a pro každý vrchol v reprezentující proměnnou (označme ji x) ověříme, zda je x vázána nějakým kvantifikátorem. Pokud vázána je, pak se na cestě od vrcholu v ke kořeni nutně musí vyskytovat vrchol w takový, že reprezentuje kvantifikátor, který váže proměnnou x. Pokud takový vrchol w neexistuje, jedná se o volný výskyt proměnné x, a proto musíme zavést nový existenční kvantifikátor k_y pro proměnnou y. Kvantifikátor k_y se potom stane novým kořenem syntaktického stromu.



Obrázek 3: Příklad zavedení existenčního uzávěru

3.2.3.2 Eliminace nadbytečných kvantifikátorů

Procházíme syntaktickým stromem do šířky. Pro každý vrchol v reprezentující kvantifikátor (označme jej k_x) vázající proměnnou (označme ji x) zkontrolujeme, zda podstrom vrcholu v obsahuje alespoň jeden vrchol reprezentující proměnnou x, která je kvantifikátorem k_x vázána. Pokud žádný takový vrchol neexistuje, kvantifikátor k_x ze syntaktického stromu odstraníme.

3.2.3.3 Přejmenování proměnných

Na začátku vytvoříme prázdnou tabulku pro evidenci použitých jmen proměnných. Dále je vhodné si uvědomit, že díky dvěma předešlým krokům nemohou nastat tyto situace:

1. Ve formuli existuje volná proměnná

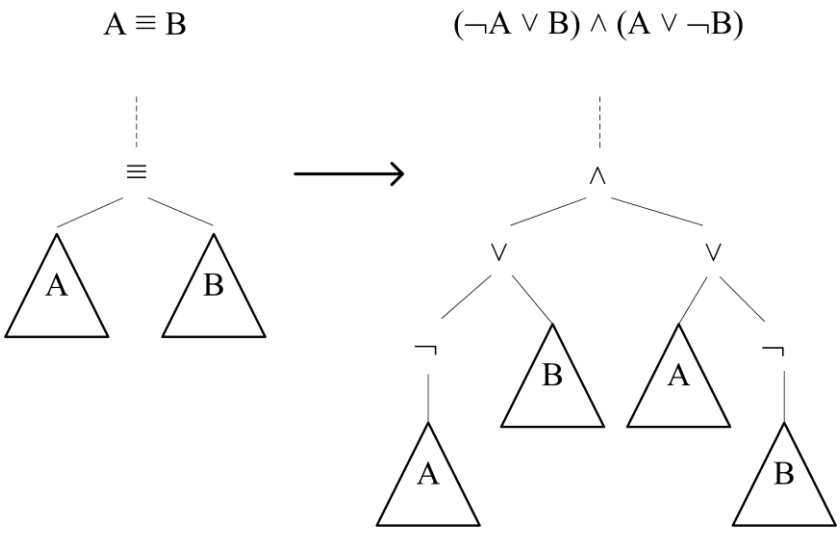
2. V podstromu vrcholu *v* reprezentujícího kvantifikátor (označme jej k_x) vázající proměnnou (označme ji *x*) se nevyskytuje žádný vrchol reprezentující proměnnou *x*.

Algoritmus realizujeme průchodem syntaktickým stromem do šířky. Pro každý vrchol *v* reprezentující kvantifikátor (označme jej k_x) vázající proměnnou (označme ji *x*) ověříme, zda proměnná *x* je již v tabulce, nebo není. Pokud není, vložíme proměnnou *x* do tabulky. Pokud ovšem je, musíme proměnnou *x* přejmenovat jak v případě vrcholu *v*, tak v celém podstromu, jehož je *v* kořen. Nakonec přejmenovanou proměnnou *x* přidáme do tabulky pro evidenci proměnných.

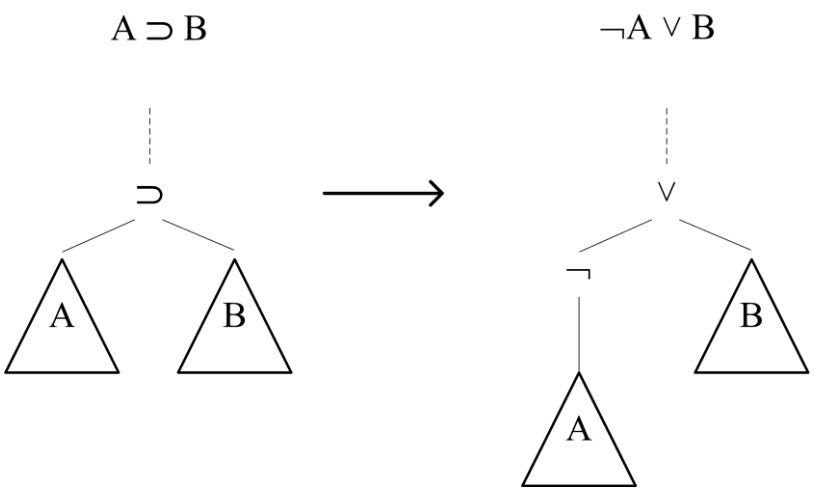
3.2.3.4 Eliminace spojek ≡, ⊃, ↑, ↓

Procházíme syntaktickým stromem do šířky a každý vrchol reprezentující spojky ≡, ⊃, ↑ a ↓ nahradíme dle níže uvedených vzorů za pomocí disjunkcí a konjunkcí.

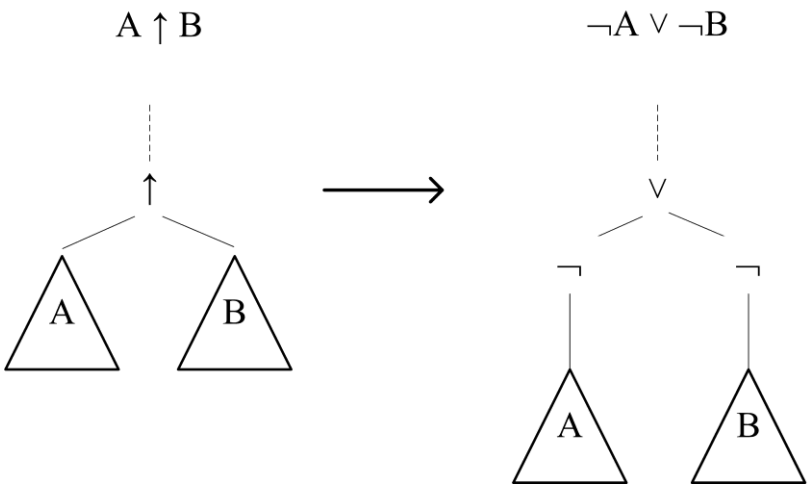
Pro eliminaci implikace budeme používat vzorec $A \equiv B \Leftrightarrow (\neg A \vee B) \wedge (A \vee \neg B)$, jelikož z hlediska kroku osmého nám dává výstup v klauzulární formě. Navíc oproti vzorci, který převádí ekvivalenci na konjunkci dvou implikací, se nám nebude do výsledného řetězce zanášet informace, že byly následně provedeny dvě operace eliminace vzniklých implikací. Tedy nebude vznikat situace, kdy by ke každé eliminaci ekvivalence nutně následovaly dvě operace eliminace implikací. V řetězci složitosti by pak hodnota „1“ na pozici 4.1 automaticky znamenala hodnotu „1“ i na pozici 4.2. V tomto případě bude hodnota jedna na pozici 4.1 a na pozici 4.2 bude hodnota nula.



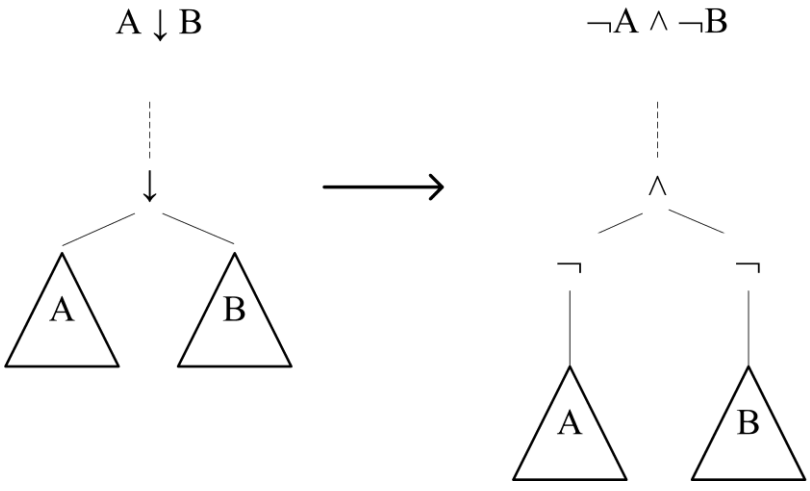
Obrázek 4: Příklad eliminace ekvivalence



Obrázek 5: Příklad eliminace implikace



Obrázek 6: Příklad eliminace spojky NAND



Obrázek 7: Příklad eliminace spojky NOR

3.2.3.5 Přesun negace dovnitř

Algoritmus pro přesun negace dovnitř bude následující. Nalezneme vrchol negace *n*, jehož přímým potomkem není vrchol reprezentující predikát a je nejblíže kořeni. Pak mohou nastat právě 4 případy a dle konkrétního vybereme příslušné pravidlo z níže uvedených, které na vrchol negace *n* aplikujeme. Znovu nalezneme vrchol negace, jehož přímým potomkem není vrchol predikátu, a je nejblíže kořeni a algoritmus opakujeme, dokud nenalezneme žádnou negaci, která by neměla jako přímého potomka vrchol predikát.

U přesunu negace dovnitř je potřeba mít na paměti, že z hlediska počtu přesunů, které bude potřeba provést, záleží na pořadí, ve kterém jednotlivé negace přesunujeme. Z hlediska výsledné formule na pořadí sice nezáleží, ale je vhodné použít optimální přístup, tedy takový, který se, mimo jiné, jednoznačně projeví na hodnotách složek řetězce složitosti. Minimální počet přesunů lze zajistit postupem, ve kterém vždy přesuneme negaci, která je nejblíže kořeni. Pokud bychom tak neučinili a zvolili jiný postup, mohla by například nastat situace, ve které bychom místo eliminace dvojí negace tuto dvojí negaci rozdělili. To by mělo za následek, že bychom museli samostatně přesunovat obě negace z oné rozdělené dvojí negace. Nakonec by se tyto dvě negace opět spojily do dvojí negace a bylo by potřeba ji eliminovat. Problémem by ale bylo, že jejich přesuny došlo ke zbytečně velkému počtu kroků, kterému bylo možno předejít. Ukažme si nyní vše na následujícím příkladu a formuli $\forall x \neg \exists y \neg[A(x) \vee B(y)]$:

Příklad postupu, ve kterém jsme provedli zbytečně mnoho přesunu negace:

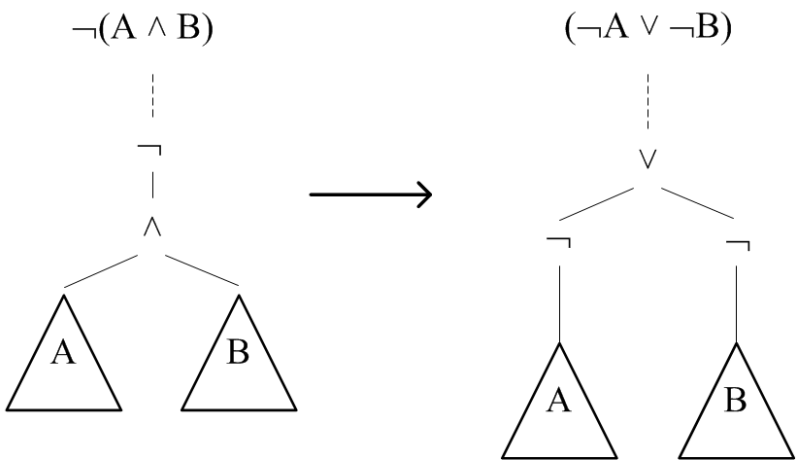
$$\forall x \neg \exists y \neg[A(x) \vee B(y)] \Leftrightarrow \forall x \forall y \neg\neg[A(x) \vee B(y)] \Leftrightarrow$$
$$\forall x \forall y \neg[\neg A(x) \wedge \neg B(y)] \Leftrightarrow \forall x \forall y [\neg\neg A(x) \vee \neg\neg B(y)] \Leftrightarrow$$
$$\forall x \forall y [A(x) \vee B(y)]$$

Postup, jenž je z hlediska počtu přesunů negace, optimální:

$$\forall x \neg \exists y \neg[A(x) \vee B(y)] \Leftrightarrow$$
$$\forall x \forall y \neg\neg[A(x) \vee B(y)] \Leftrightarrow \forall x \forall y [A(x) \vee B(y)]$$

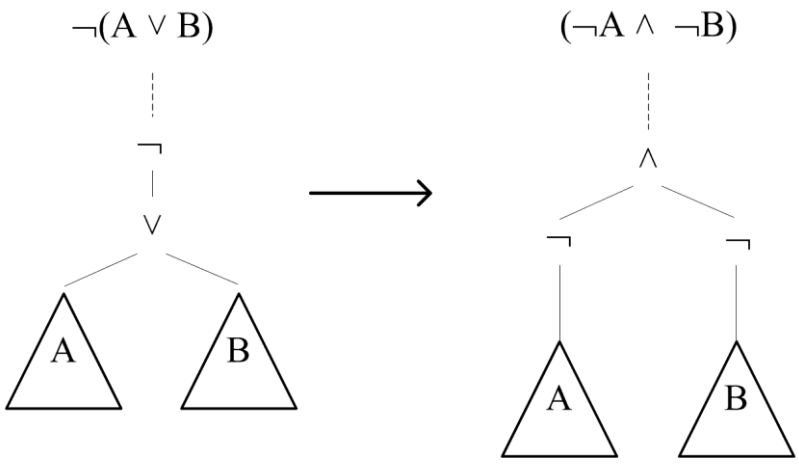
Situace, které mohou nastat u negací, které nemají jako přímého potomka predikát, jsou tyto:

1. Negace se váže na druhou negaci, aplikujeme pravidlo dvojí negace: $\neg\neg A = A$. Aplikace tohoto pravidla se nijak neprojeví v řetězci složitosti, jelikož eliminace dvojí negace není v algoritmu určení složitosti úlohy zachycena.
2. Negace se váže na konjunkci, aplikujeme De Morganovy zákony: $\neg(A \wedge B) \Leftrightarrow (\neg A \vee \neg B)$. U každého nalezeného vzoru $\neg(A \wedge B)$ je nutné ještě prozkoumat, zda jsou výrazy A, B literály či nikoliv, abychom mohli správně klasifikovat, zda se nejedná o speciální případy kroku číslo 5.1, tedy o kroky 5.1.1, nebo 5.1.2.



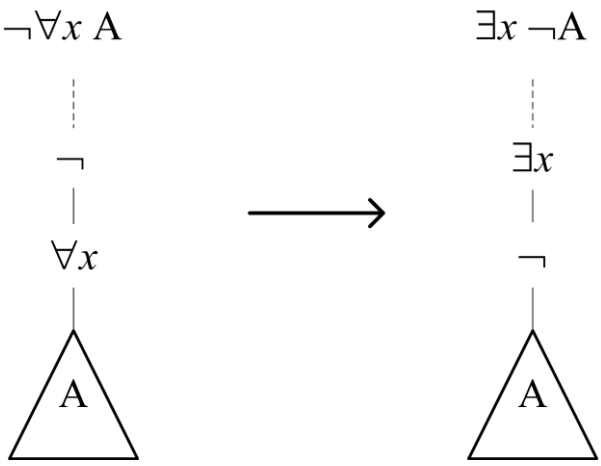
Obrázek 8: Schéma eliminace negace před konjunkcí

3. Negace se váže na disjunkci, aplikujeme De Morganovy zákony: $\neg(A \vee B) \Leftrightarrow (\neg A \wedge \neg B)$. U každého nalezeného vzoru $\neg(A \vee B)$ je nutné opět prozkoumat, zda jsou výrazy A, B literály či nikoliv, abychom mohli správně klasifikovat, zda se nejedná o speciální případy kroku číslo 5.1, tedy o kroky 5.1.1, nebo 5.1.2.



Obrázek 9: Schéma eliminace negace před disjunkcí

4. Negace se váže na kvantifikátor. Aplikujeme De Morganovy zákony: $\neg\forall x\, A \Leftrightarrow \exists x\, \neg A$, $\neg\exists x\, A \Leftrightarrow \forall x\, \neg A$



Obrázek 10: Příklad přesunu negace za všeobecný kvantifikátor

3.2.3.6 Přesun kvantifikátorů doprava

Pro každý kvantifikátor k_x v pořadí od nejvzdálenějšího od kořene syntaktického stromu zkontrolujeme, zda potomkem kvantifikátoru k, je nějaký predikát p. Pokud je, pak pokračujeme dalším kvantifikátorem. Pokud není, mohou nastat tyto dvě situace:

1. Následovníkem k_x je jiný kvantifikátor l, což znamená, že l jsme už prošetřili (je dále od kořene stromu) a l nebylo možné více doprava přesunout. Pokud je l stejný kvantifikátor jako k_x (oba jsou existenční nebo jsou oba všeobecné), pak můžeme k_x s l prohodit a pokračujeme znovu od bodu jedna pro kvantifikátor k_x. Pokud je l jiný kvantifikátor než k_x, pak k_x přesunout nemůžeme a pokračujeme dalším kvantifikátorem.

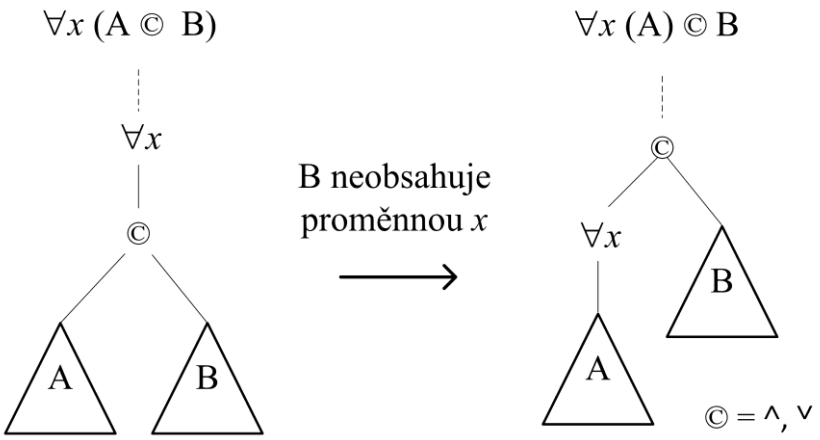
   Příklad:

   $$\forall x \forall y\, [\mathrm{P}(x, y) \wedge \exists z\, \mathrm{Q}(y, z)] \Leftrightarrow \forall y \forall x\, [\mathrm{P}(x, y) \wedge \exists z\, \mathrm{Q}(y, z)] \Leftrightarrow$$
   $$\forall y\, [\forall x\, \mathrm{P}(x, y) \wedge \exists z\, \mathrm{Q}(y, z)]$$

   Všeobecný kvantifikátor proměnné y nelze přes konjunkci přesunout. Potomkem všeobecného kvantifikátoru proměnné x, je všeobecný kvantifikátor proměnné y. Tyto kvantifikátory můžeme tedy prohodit, což jsme učinili prvním krokem v příkladu a získali tak formuli $\forall y \forall x\, [\mathrm{P}(x, y) \wedge \exists z\, \mathrm{Q}(y, z)]$. Nyní můžeme všeobecný kvantifikátor proměnné x, přesunout přes konjunkci, protože proměnná x se vyskytuje pouze v jednom jejím podstromě. Přesun všeobecného kvantifikátoru proměnné x jsme učinili ve druhém kroku příkladu.

2. Následovníkem k je spojka konjunkce, nebo disjunkce s (ostatní jsme už eliminovali). V tomto případě prozkoumáme oba podstromy spojky s, zda se v nich vyskytuje proměnná x, kterou k_x váže. V případě, že se proměnná x vyskytuje pouze v jednom podstromu, potom přesuneme kvantifikátor k_x za spojku s tak, že se stane vrcholem

podstromu, ve kterém se vyskytuje proměnná x. Rekurzivně postup pro k_x opakujeme, dokud jej lze přesunovat.

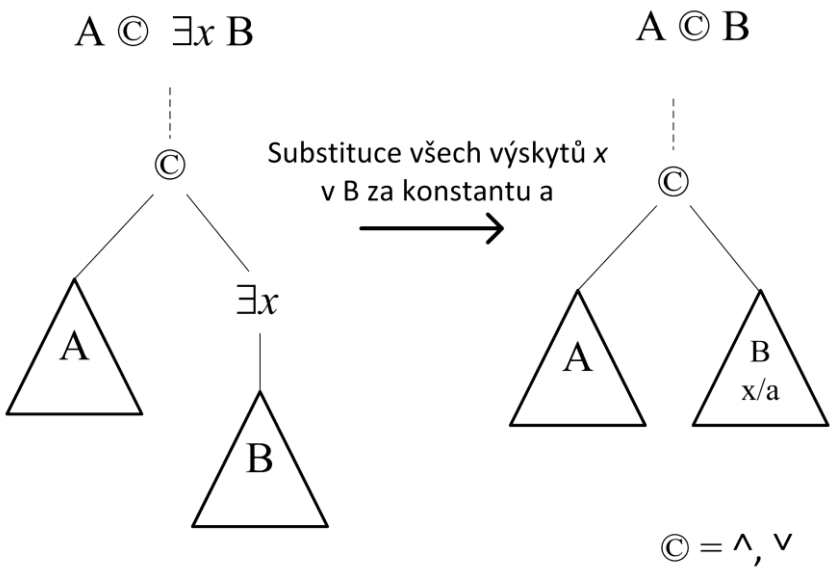


Obrázek 11: Příklad přesunu kvantifikátoru doprava

3.2.3.7 Skolemizace funkcí a konstantou

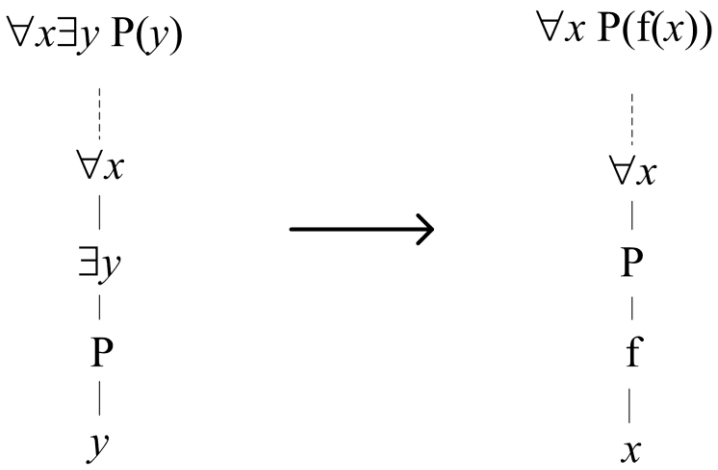
Tento krok je rozdělen na dvě části, Skolemizace konstantou a Skolemizace funkcí, které se budou řešit v tomto pořadí:

1. Procházíme syntaktickým stromem do hloubky a pro každý kvantifikátor k_x postupujeme následovně.
    a. Pokud je k existenční kvantifikátor, pak musíme v podstromu kvantifikátoru k_x, nahradit všechny výskyty proměnné x, kterou kvantifikátor k_x váže, nově zavedenou Skolemovou konstantou. Nakonec kvantifikátor k_x ze syntaktického stromu odstraníme a pokračujeme v průchodu do hloubky.
    b. Pokud je k_x všeobecný kvantifikátor, pak v jeho podstromu již není možné zavést jakoukoliv Skolemovu konstantu (je možné zavést jedině Skolemovy funkce), a proto průchod do hloubky pro tuto větev ukončíme a pokračujeme průchodem další větví syntaktického stromu.



Obrázek 12: Příklad Skolemizace konstantou

2. Procházíme syntaktickým stromem do šířky a nalezneme všechny zbývající existenční kvantifikátory k_x. Pro každý takový kvantifikátor k_x, nalezneme všechny všeobecné kvantifikátory $k_{y1}...k_{yn}$ na cestě od k_x ke kořeni stromu. Nakonec všechny výskyty proměnné x vázané kvantifikátorem k_x nahradíme nově zavedenou Skolemovou funkcí $f_k(y_1...y_2)$, kde $y_1...y_2$ jsou proměnné vázané kvantifikátory $k_{y1}...k_{yn}$ a odstraníme kvantifikátor k_x ze syntaktického stromu.



Obrázek 13: Příklad Skolemizace funkcí

3.2.3.8 Přesun kvantifikátorů doprava

Algoritmus budeme realizovat průchodem syntaktickým stromem do šířky. Každý vrchol reprezentující kvantifikátor přesuneme tak, že se z něj stane vrchol syntaktického stromu. Tímto může dojít k záměně pořadí kvantifikátorů ve formuli, což si můžeme dovolit z důvodu, že ve formuli jsou již jen všeobecné kvantifikátory, na jejichž pořadí nezáleží.

3.2.3.9 Převedení formule na klauzulární tvar - uplatnění distributivních pravidel

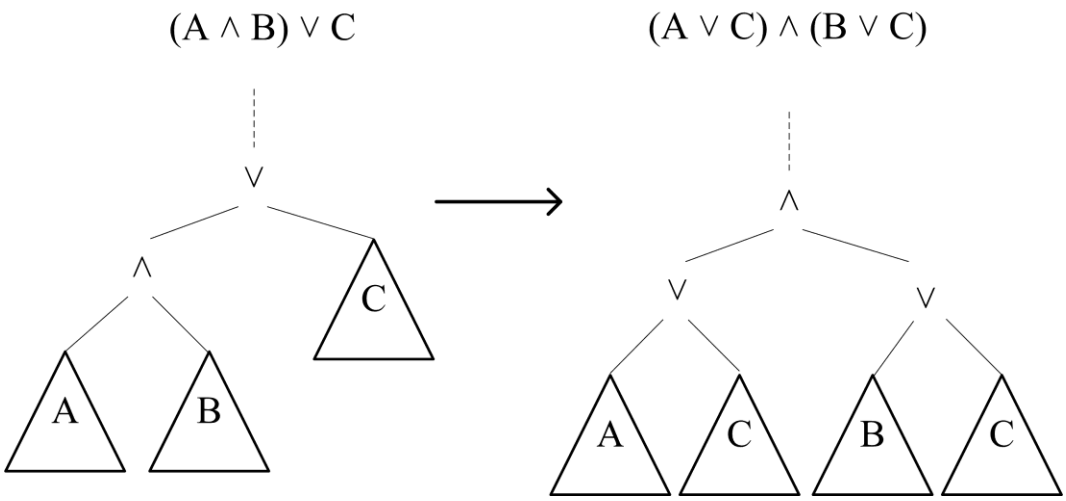
Před popisem algoritmu si musíme uvědomit následující skutečnost. Budeme-li uplatňovat distributivní pravidla na speciální případ, odpovídající pozici 9.4 řetězce složitosti, tedy alespoň jeden z výrazů A, B, C je složený a nelze jej popsat jinak a výraz C obsahuje alespoň jednu konjunkci, pak není jednoznačně dáno, který z výrazu $(A \wedge B)$, C máme distribuovat. Pro studenta je za takové situace obecně jednodušší distribuovat výraz s menším počtem literálů. Tento postup rovněž vede v dalším průběhu k nutnosti uplatňovat distributivní pravidla na jednodušší výrazy.

Příklad:

$$(A \wedge B) \vee (C \wedge (E \wedge F)) \Leftrightarrow$$
$$[A \vee (C \wedge (E \wedge F))] \wedge [B \vee (C \wedge (E \wedge F))] \Leftrightarrow \ldots$$

V příkladu jsme distribuovali výraz $(A \wedge B)$, jelikož obsahuje méně literálů než výraz $(C \wedge (E \wedge F))$.

Procházíme syntaktickým stromem a hledáme vrchol disjunkce, který je nejvzdálenější od kořene stromu a zároveň má jako přímého potomka alespoň jeden vrchol konjunkce. Tímto nalezneme výraz $(A \wedge B) \vee C$, u kterého ještě musíme analyzovat strukturu jednotlivých výrazů A, B a C. Na základě analýzy struktury výrazů A, B a C rozhodneme, zda se nejedná o speciální případ kroku devátého, tedy kroky 9.1 až 9.4. Pokud ano, pak musíme ještě rozhodnout, o který konkrétní speciální případ se jedná. Nakonec na nalezený výraz $(A \wedge B) \vee C$ aplikujeme distributivní zákon dle obrázku.



Obrázek 14: Příklad uplatnění distributivního zákona na vzor (A ∧ B) ∨ C

Tento postup provádíme, dokud při průchodu stromem, již nenalezneme žádnou disjunkci, která je následována konjunkcí.

4 Vzory pro Skolemizaci

Ve druhé části této práce se budeme zabývat návrhem systému pro generování vzorů pro skolemizaci na základě řetězce složitosti. Vstupem sytému bude tedy řetězec složitosti, jehož hodnoty na jednotlivých složkách odpovídají krokům algoritmu skolemizace a hierarchickému řádu formule (vzoru), což bylo podrobně popsáno v předchozí kapitole. Výstupem systému bude vzor, z něhož se vytvoří jedna, nebo více formulí PL1. Řetězec složitosti formulí na výstupu bude shodný s řetězcem složitosti, jenž byl na vstupu.

Systém bude generovat konečný počet vzorů. U každého vzoru je vypočítána jeho složitost, jak z hlediska hierarchického řádu, tak i složitosti aplikace algoritmu skolemizace. Maximální hierarchický řád, který bude možné generovat, omezíme na hodnotu 3. Pro účely výuky nemá smysl vytvářet příliš složité vzory ke skolemizaci, jelikož při jejich řešení se studenti přestanou orientovat ve struktuře formule a chyby v jednotlivých krocích algoritmu skolemizace jsou potom především způsobeny ztrátou orientace a nikoliv neznalostí algoritmu skolemizace.

V následujících částech budeme používat metasymboly @, $, + a *, které budou mít tento význam:

- @ je metasymbol $\wedge$, $\vee$, $\supset$, nebo $\equiv$.
- $ je metasymbol $\wedge$, $\vee$, nebo $\supset$.
- + je metasymbol $\wedge$, $\vee$, nebo $\equiv$.
- * je metasymbol $\wedge$, nebo $\vee$.

4.1 Vymezení podmnožiny jazyka PL1

Pro účely systému pro generování vzorů ke skolemizaci vymezím z praktických důvodů podmnožinu jazyka PL1, kterou bude systém generovat.

Systém tedy bude generovat podmnožinou jazyka PL1 splňující tato omezení:

- Z logických spojek budou použity negace, implikace, ekvivalence, konjunkce a disjunkce.
- Predikáty budou *n*-ární, kdy $n = 1, 2, 3, 4$.
- Funkce budou *n*-ární, kdy $n = 0, 1$.
- Argumentem funkce bude pouze proměnná, nebo konstanta.
- Názvy predikátů budou velká písmena, případně následována sérií čísel, např. A, B, A123.
- Názvy funkcí budou malá písmena *f* až *m*, případně následována sérií čísel.
- Názvy konstant (nulárních funkcí) budou malá písmena *a* až *d*, případně následována sérií čísel.

- Názvy proměnných budou malá písmena *p* až *z*, případně následována sérií čísel.
- V případě opakovaného použití logické spojky bude potřeba formuli plně uzávorkovat dle definice, např. $((A \supset B) \supset C)$.
- Maximální hierarchický řád generovaných formulí bude roven hodnotě 3.

4.2 Vymezení pojmu vzor

V této kapitole budeme často používat pojem vzor. Vzor by měl mít takovou vlastnost, že má jednoznačně určený řetězec složitosti. Co to ale vlastně vzor je? Pokusme se jej nyní co možná nejpřesněji definovat. Vztah mezi pojmy vzor a formule je tento:

- Každému vzoru odpovídá vždy alespoň jedna formule.
- Každé formuli odpovídá právě jeden vzor.

Pojmem vzor tedy budeme rozumět množinu formulí PL1, která má následující vlastnosti:

1. Na jménech predikátů, funkcí a proměnných nezáleží. Mějme formule A, B. Existuje-li bijekce *f* mezi jmény predikátu, funkcí a proměnných formulí B, A a formule B' vzniklá aplikací *f* na jména všech predikátů, funkcí a proměnných formule B, je stejná jako formule A, potom formule A i B jsou tím stejným vzorem.

   Příklad:

   $$A = \exists x \forall y\, [P(x) \wedge Q(y)]$$
   $$B = \exists u \forall v\, [R(u) \wedge S(v)]$$
   $$f(R) = P,\ f(S) = Q,\ f(u) = x,\ f(v) = y$$
   $$B' = \exists x \forall y\, [P(x) \wedge Q(y)] = A$$

   pak A, B jsou stejným vzorem

   Zobrazení *f* je zjevně bijekce, a proto formule A i B jsou stejným vzorem.

2. Obsahuje-li formule A výraz B, takový, že všechny binární spojky ve výrazu B jsou stejné a zároveň jsou komutativní a asociativní (ve vymezené podmnožině jazyka PL1 jsou to spojky $\wedge$, $\vee$, $\equiv$), pak všechny formule A' vzniklé aplikací komutativního, asociativního zákona a jejich kombinací na všechny možné podmnožiny binárních spojek výrazu B jsou stejným vzorem.

   Příklad:

   $$B = A = \exists x\, [P(x) \wedge \neg \forall y\, Q(y)]$$
   $$A' = \exists x\, [\neg \forall y\, Q(y) \wedge P(x)]$$

Výraz B obsahuje jednu binární asociativní a komutativní spojku ($\wedge$). Aplikací komutativního zákona na výraz B vznikne formule A', a proto jsou formule A, A' stejným vzorem.

3. Na pořadí termů u predikátu nezáleží.

   Příklad:

   $$A = \exists x \forall y\, P(x, y, y)$$
   $$B = \exists x \forall y\, P(x, y, x)$$
   $$C = \exists x \forall y\, P(y, y, x)$$

   Formule A, B a C jsou stejným vzorem, protože se liší pouze pořadím termů u predikátu P.

4. Na pořadí kvantifikování proměnných nezáleží u všech kvantifikátorů, které ve svém podstromu neobsahují žádnou binární spojku a negace se vyskytují pouze bezprostředně před predikátem.

   Příklad:

   $$A = \exists x \forall y\, \neg P(x, y)$$
   $$B = \exists y \forall x\, \neg P(x, y)$$

   Formule A, B se liší pouze pořadím, v jakém jsou kvantifikovány proměnné x a y. Formule A, B jsou tedy stejným vzorem.

5. Na pořadí stejných kvantifikátorů, které se v syntaktickém stromě vyskytují bezprostředně pod sebou, nezáleží.

   Příklad:

   $$A = \forall y \exists x \exists z\, [P(x, z) \wedge Q(y)]$$
   $$A = \forall y \exists z \exists x\, [P(x, z) \wedge Q(y)]$$

   Formule A, B se liší pouze pořadím kvantifikátorů $\exists x$ a $\exists z$. Formule A, B jsou tedy stejným vzorem.

4.3 Popis systému pro generování vzorů

Máme-li na vstupu konkrétní formuli, výpočet jejího řetězce složitosti převedení do klauzulární Skolemovy formy je poměrně snadný, což jsme ukázali v kapitole třetí. Nicméně řešení opačného problému, tedy takového, kdy na vstupu bude řetězec složitosti a úkolem je nalézt formuli takovou, že její řetězec složitosti převedení do Skolemovy klauzulární formy odpovídá řetězci složitosti na vstupu, je v obecném případě obtížné. Z tohoto důvodu nebude navržený

systém fungovat na principu takovém, že vygeneruje přímo ke konkrétnímu řetězci složitosti množinu vzorů, ale vzory budou před použitím systému vygenerovány a systém z nich pak bude podle vstupu od uživatele vybírat.

Na nejvyšší úrovni abstrakce bude tedy systém pracovat následovně:

1. Vygeneruje množinu vzorů pro příslušný hierarchický řád.
2. Pro každý vzor vypočítá jeho řetězec složitosti převedení do Skolemovy klauzulární formy.
3. Eliminují se duplicitní vzory.
4. Každý vzor se společně s řetězcem složitosti uloží do persistentního datového úložiště, vytvoří tak databází vzorů, u nichž je známý jejich řetězec složitosti. Tyto čtyři kroky budeme dále označovat jako fáze předgenerování.
5. Při každém dalším spuštění již bude systém načítat vygenerované vzory z persistentního datového úložiště.
6. Pro konkrétní vstup od uživatele, tedy řetězec složitosti, vybere z databáze vzorů takové, jejichž řetězec složitosti bude odpovídat řetězci složitosti na vstupu.
7. Pro vybraný vzor bude uživateli zobrazena množina formulí, které lze z vybraného vzoru získat substitucemi jmen predikátů, nebo změnou struktury formulí bez změny řetězce složitosti, viz vymezení pojmu vzor.

Pro komfort uživatele bude systém dále umožňovat:

1. V případě, že systém nenalezne v databázi žádný vzor s řetězcem složitosti shodným se vstupem od uživatele, nabídne uživateli řetězce složitosti lišící se v nejmenším možném počtu pozic takové, pro které existují v databázi příslušné vzory.
2. Systém umožní pro jednotlivé vzory ukázat průběh převedení vzoru do Skolemovy klauzulární formy po jednotlivých devíti krocích.
3. Systém umožní exportovat databázi vzorů zvoleného hierarchického řádu společně s řetězci složitosti do textového souboru.

U řetězce složitosti nebudeme v rámci systému pro generování vzorů uvažovat složku hierarchický řád formule po převedení do Skolemovy klauzulární formy. Tzn., že uživatel nebude mít možnost vybírat z databáze vzorů podle hierarchického řádu formule po převedení do Skolemovy klauzulární formy. Pokud bychom tuto složku chtěli do budoucna uvažovat, princip fungování sytému by se vůbec nezměnil. Pouze by bylo potřeba omezit maximální možný hierarchický řád formule po skolemizaci a hodnotu hierarchického řádu v binární reprezentaci přidat k řetězci složitosti.

V následující tabulce uvedeme pro přehled počet různých předgenerovaných vzorů pro jednotlivé hierarchické řády.

Tabulka 2: Orientační počet vygenerovaných vzorů

Hierarchický řád	Počet různých vygenerovaných vzorů
0	72
1	3 233
2	182 310
3	1 119 894

4.3.1 Kódování řetězce složitosti a způsob ukládání

Všechny složky řetězce složitosti nabývají, až na dvě výjimky (hierarchický řád vstupní formule a hierarchický řád formule po převedení do Skolemovy klauzulární formy, který zatím neuvažujeme), pouze dvou hodnot. Řetězec složitosti tak vlastně odpovídá binárnímu zápisu celého čísla, což je vhodný způsob, jak řetězec efektivně zakódovat. Vzhledem k počtu složek řetězce složitosti budeme používat 32-bitové číslo, nejméně významnému bitu bude odpovídat poslední složka řetězce složitosti. Zbývá vyřešit, jak efektivně zakódovat hierarchický řád vstupní formule. Tento problém lze řešit dvěma způsoby:

1. Jelikož hierarchický řád ve vymezené podmnožině jazyka PL1 nabývá hodnot od 0 do 3, můžeme jeho hodnotu zapsat pomocí dvojice bitů.
2. Pro každý hierarchický řád vytvořit zvlášť soubor, ve kterém budou uloženy jen vzory daného hierarchického řádu. V případě použití SŘBD jakožto persistentního datového úložiště toto odpovídá vytvoření jedné tabulky pro každý hierarchický řád. Toto řešení je z důvodu menší velikosti vzniklých souborů (tabulek) efektivnější.

V navrženém systému použijeme druhý způsob řešení, systém pak bude vytvářet zvláštní soubory pro jednotlivé hierarchické řády. Za persistentní datové úložiště zvolíme textové soubory. V případě použití SŘBD by princip fungování systému zůstal nezměněn. Pouze by bylo potřeba implementovat komunikací se zvoleným SŘBD. Hlavním důvodem použití textových souborů místo SŘBD, je možnost jednoduše použít takovouto aplikaci na běžné uživatelské stanici. Aplikace je pro komfort uživatele navržená jako desktopová a v případě použití SŘBD by bylo nutné mít na klientské stanici instalováno příslušné SŘBD. Komunikace se SŘBD přes síť by znamenala vzhledem k velkému objemu dat velice pomalou odezvu systému. Textový soubor, namísto binárního, volíme z důvodu možnosti jeho komfortního prohlížení.

Struktura textového souboru bude následující.

1. Každý vzor společně s číslem kódujícím řetězec složitosti bude uložen na novém řádku.
2. Oddělovačem vzoru od čísla kódujícího řetězec složitosti bude středník.
3. Číslo kódující řetězec složitosti bude zapsáno v desítkové soustavě.

Příklad řádku textového datového souboru:

1z((1x1yA(x,y) * B(u)) > (1vC(v) <> D(z)));1517233

4.3.2 Fáze generování vzorů

Generování vzorů budeme realizovat po krocích (fázích), které budou v principu pro všechny hierarchické řády shodné, včetně jejich pořadí. Vstupem do první fáze generování bude strukturní hierarchický vzor, což je vzor, který neobsahuje kvantifikátory a všechny jeho predikáty jsou nulární.

Příklad hierarchického vzoru (A-F jsou nulární predikáty):

$$(A \vee B) \wedge [(C \supset D) \wedge (E \equiv F)]$$

Pro každý hierarchický řád musíme tedy předem definovat množinu strukturních vzorů, ze kterých systém vygeneruje vzory pro skolemizaci. Strukturní vzory volíme zejména s ohledem na didaktické zaměření systému. Není vhodné, aby systém generoval vzory, při jejichž převádění do Skolemovy klauzulární formy narůstá hierarchický řád převáděné formule, a tudíž se příklad stává pro studenta velmi nepřehledným. Pro účely rozhodování, zda konkrétní strukturní vzor zařadit do množiny, ze které se budou generovat výsledné vzory, se nabízí možnost použít systém popsaný v kapitole 3, který vypočítá řetězec složitosti převedení konkrétního strukturního vzoru do Skolemovy klauzulární formy a zároveň vrátí formuli vzniklou aplikací algoritmu skolemizace. Součástí této práce je i volba vhodných strukturních vzorů.

Jednotlivými fázemi generování, které se provedou pro každý strukturní vzor daného hierarchického řádu, budou:

1. Generování proměnných.
2. Generování kvantifikátorů.
3. Generování vzorů vyžadujících zavedení existenčního kvantifikátoru.

4.3.2.1 Generování proměnných

V této fázi bude systém k predikátům strukturního vzoru na vstupu generovat různé kombinace proměnných. K jednomu strukturnímu vzoru na vstupu tak vznikne množina vzorů, doplněných o proměnné, která bude vstupem do další fáze. Postup generování proměnných bude prakticky totožný pro všechny hierarchické řády s výjimkou nultého hierarchického řádu. Pro ostatní hierarchické řády bude generování proměnných probíhat následujícím způsobem:

1. Z konfiguračního souboru načteme pro daný strukturní vzor všechna pravidla pro generování proměnných do predikátů. V rámci této práce budeme pravidla zapisovat jako řadu závorkových dvojic (,). Každá závorková dvojice bude odpovídat predikátu ve strukturním vzoru. V závorkových dvojicích pak budou proměnné, které se budou do odpovídajících predikátů vkládat.
2. Pro každé pravidlo načtené v předešlém kroku vytvoříme vzor, který přidáme do množiny, která bude výstupem fáze generování proměnných.

Příklad: Ze strukturního vzoru $(A \vee B) \wedge C$ *a pravidla (x, y) (x, z) (x) vznikne vzor:*

$$[A(x, y) \vee B(x, z)] \wedge C(x)$$

Pravidla pro generování proměnných budou zapsána v konfiguračním souboru a jejich budoucí změna bude možná bez nutnosti jakkoliv zasahovat do zdrojového kódu systému. Součástí této práce je i volba vhodných pravidel pro generování proměnných.

4.3.2.2 Generování kvantifikátorů

Vstupem druhé fáze generování bude množina vzorů z fáze první, tedy generování proměnných. Pro každý vzor na vstupu systém vytvoří novou množinu vzorů, ve kterých budou všechny proměnné kvantifikovány vygenerovanými kvantifikátory. Výstupem bude sjednocení takto vzniklých množin vzorů. Konkrétní realizace generování kvantifikátorů bude specifická pro jednotlivé hierarchické řády. V zásadě bude generování kvantifikátorů pevně dáno a nebude jej možné jakkoliv nastavovat v konfiguračním souboru. Generovat kvantifikátory budeme obecně těmito způsoby.

4.3.2.2.1 Kvantifikátory před jednotlivými predikáty vzoru

Kvantifikátory budou přímo před jednotlivými predikáty daného vzoru. Počet predikátu ve vzoru označíme *n* a postupovat budeme následovně:

1. Pro každý predikát p_i $i \in \{1..n\}$ z daného vzoru vygenerujeme množinu variací kvantifikátorů s opakováním z množiny {všeobecný kvantifikátor, existenční kvantifikátor} délky shodné s počtem různých proměnných u predikátu p_i daného vzoru. Tuto množinu variací kvantifikátorů označíme V_i.
2. Vytvoříme množinu V jako kartézský součin V_1 až V_n. $V = V_1 \times V_2 \times \ldots \times V_n$
3. V případě, že ve vzoru existuje výraz B takový, že všechny binární spojky v B jsou stejné, komutativní a negace se nacházejí pouze před predikáty, pak musíme některé prvky z množiny *V* odstranit, abychom zabránili vygenerování více stejných vzorů. Odebírat budeme, tak aby platilo:

   $$\forall p_i, \forall p_j \in B, i \neq j: [(\ldots, V_i, \ldots, V_j, \ldots) \in V \wedge |V_i| = |V_j|] \supset (\ldots, V_j, \ldots, V_i, \ldots) \notin V$$

   Podmínkou $|V_i| = |V_j|$ vyjadřujeme, že počet variací v množině V_j je shodný s počtem variací v množině V_i. Počet variací *k*-té třídy s opakováním z *n* prvků je roven n^k. [5] Z podmínky na rovnost mohutností množin $|V_i|$ a $|V_j|$ tak vyplývá, že délka sekvencí kvantifikátorů (třída variací), tedy prvků množin V_j a V_i, musí být shodná.

   Tento krok může být vypuštěn, jelikož duplicitní vzory budou eliminovány během fáze eliminace duplicitních vzorů. Z důvodu větší efektivity při jeho použití jej ale raději uvádíme.

4. Pro každou *n*-tici *t* z množiny *V* musíme nyní vytvořit nový vzor tak, že variace kvantifikátorů na *i*-té pozici *n*-tice *t* se vloží před *i*-tý predikát vzoru. Na pořadí

kvantifikování proměnných z definice vzoru nezáleží, a proto bude pořadí proměnných u kvantifikátorů vždy shodné, jako je u predikátu, před kterým se kvantifikátory nacházejí.

Příklad: Ze vzoru A(x, y) ∧ B(x, z) vygenerujeme dle výše uvedeného algoritmu tyto vzory:

$$\forall x \forall y\ \mathrm{A}(x, y) \wedge \forall x \forall z\ \mathrm{B}(x, z)$$
$$\forall x \forall y\ \mathrm{A}(x, y) \wedge \forall x \exists z\ \mathrm{B}(x, z)$$
$$\forall x \forall y\ \mathrm{A}(x, y) \wedge \exists x \forall z\ \mathrm{B}(x, z)$$
$$\forall x \forall y\ \mathrm{A}(x, y) \wedge \exists x \exists z\ \mathrm{B}(x, z)$$
$$\sout{\forall x \exists y\ \mathrm{A}(x, y) \wedge \forall x \forall z\ \mathrm{B}(x, z)}$$
$$\forall x \exists y\ \mathrm{A}(x, y) \wedge \forall x \exists z\ \mathrm{B}(x, z)$$
$$\forall x \exists y\ \mathrm{A}(x, y) \wedge \exists x \forall z\ \mathrm{B}(x, z)$$
$$\forall x \exists y\ \mathrm{A}(x, y) \wedge \exists x \exists z\ \mathrm{B}(x, z)$$
$$\sout{\exists x \forall y\ \mathrm{A}(x, y) \wedge \forall x \forall z\ \mathrm{B}(x, z)}$$
$$\sout{\exists x \forall y\ \mathrm{A}(x, y) \wedge \forall x \exists z\ \mathrm{B}(x, z)}$$
$$\exists x \forall y\ \mathrm{A}(x, y) \wedge \exists x \forall z\ \mathrm{B}(x, z)$$
$$\exists x \forall y\ \mathrm{A}(x, y) \wedge \exists x \exists z\ \mathrm{B}(x, z)$$
$$\sout{\exists x \exists y\ \mathrm{A}(x, y) \wedge \forall x \forall z\ \mathrm{B}(x, z)}$$
$$\sout{\exists x \exists y\ \mathrm{A}(x, y) \wedge \forall x \exists z\ \mathrm{B}(x, z)}$$
$$\sout{\exists x \exists y\ \mathrm{A}(x, y) \wedge \exists x \forall z\ \mathrm{B}(x, z)}$$
$$\exists x \exists y\ \mathrm{A}(x, y) \wedge \exists x \exists z\ \mathrm{B}(x, z)$$

Přeškrtnuté vzory jsou takové, které jsme eliminovali krokem tři.

4.3.2.2.2 Kvantifikátory na začátku vzoru

Generování kvantifikátorů na začátek vzoru bude o něco komplikovanější, jelikož v případě, že kvantifikátory nejsou přímo před predikátem, pak obecně na pořadí kvantifikování proměnných z hlediska řetězce složitosti záleží. Kvantifikátory na začátku vzoru budeme generovat následujícím způsobem:

1. Vytvoříme množinu *M*, obsahující jména proměnných, které se vyskytují ve vzoru.
2. Vygenerujeme množinu variací s opakováním z množiny {všeobecný kvantifikátor, existenční kvantifikátor} délky |*M*|, kterou označíme *V*.
3. Vygenerujeme všechny permutace množiny *M*, tím získáme všechny možné pořadí kvantifikování proměnných.
4. Pro všechny různé dvojice *v*, *m* takové, že $v \in V$ a $m \in M$ vytvoříme nový vzor, který bude mít na začátku sekvenci kvantifikátorů, odpovídající variaci *v*, a pořadí kvantifikování proměnných bude odpovídat permutaci *m*.
5. Jelikož na pořadí stejných kvantifikátorů nezáleží a předešlé kroky generují i vzory, které se liší pouze v pořadí stejných kvantifikátorů, musíme takovéto vzory eliminovat.

Příklad: Ze vzoru A(x, y) ∧ B(x, y) vygenerujeme dle výše uvedeného algoritmu tyto vzory:

$$\forall x \forall y\ [A(x, y) \wedge B(x, y)]$$
$$\forall x \exists y\ [A(x, y) \wedge B(x, y)]$$
$$\exists x \forall y\ [A(x, y) \wedge B(x, y)]$$
$$\exists x \exists y\ [A(x, y) \wedge B(x, y)]$$
~~$\forall y \forall x\ [A(x, y) \wedge B(x, y)]$~~
$$\forall y \exists x\ [A(x, y) \wedge B(x, y)]$$
$$\exists y \forall x\ [A(x, y) \wedge B(x, y)]$$
~~$\exists y \exists x\ [A(x, y) \wedge B(x, y)]$~~

Přeškrtnuté vzory jsou takové, které jsme eliminovali krokem pátým. Přesto jsme ale v příkladu vygenerovali duplicitní vzory, a to konkrétně tyto:

$$\forall x \exists y\ [A(x, y) \wedge B(x, y)]$$
$$\forall y \exists x\ [A(x, y) \wedge B(x, y)]$$

a

$$\exists x \forall y\ [A(x, y) \wedge B(x, y)]$$
$$\exists y \forall x\ [A(x, y) \wedge B(x, y)]$$

Tyto duplicity nyní řešit nebudeme, jejich eliminaci provedeme ve fázi eliminace duplicitních vzorů.

4.3.2.2.3 Kvantifikátory před závorkami

Generování kvantifikátorů před závorky, tedy před složené výrazy vzoru, bude de-facto obdobou generování kvantifikátorů na začátek vzoru. Jediným rozdílem je skutečnost, že generování kvantifikátorů před jeden výraz (závorku) musí být zcela nezávislé od generování kvantifikátorů před ostatní výrazy (závorky), abychom vygenerovali všechny možné kombinace kvantifikátorů a pořadí proměnných před jednotlivými výrazy (závorkami).

Příklad: Ze vzoru [A(x) ∨ B(y)] ∧ C(u) vygenerujeme dle výše uvedeného algoritmu tyto vzory:

$$\forall x \forall y\ [A(x) \vee B(y)] \wedge \forall u\ C(u)$$
$$\forall x \forall y\ [A(x) \vee B(y)] \wedge \exists u\ C(u)$$
$$\forall x \exists y\ [A(x) \vee B(y)] \wedge \forall u\ C(u)$$
$$\forall x \exists y\ [A(x) \vee B(y)] \wedge \exists u\ C(u)$$
$$\exists x \forall y\ [A(x) \vee B(y)] \wedge \forall u\ C(u)$$
$$\exists x \forall y\ [A(x) \vee B(y)] \wedge \exists u\ C(u)$$
$$\exists x \exists y\ [A(x) \vee B(y)] \wedge \forall u\ C(u)$$
$$\exists x \exists y\ [A(x) \vee B(y)] \wedge \exists u\ C(u)$$
~~$\forall y \forall x\ [A(x) \vee B(y)] \wedge \forall u\ C(u)$~~

$$\forall y \forall x\ [\mathrm{A}(x) \vee \mathrm{B}(y)] \wedge \exists u\ \mathrm{C}(u)$$
$$\forall y \exists x\ [\mathrm{A}(x) \vee \mathrm{B}(y)] \wedge \forall u\ \mathrm{C}(u)$$
$$\forall y \exists x\ [\mathrm{A}(x) \vee \mathrm{B}(y)] \wedge \exists u\ \mathrm{C}(u)$$
$$\exists y \forall x\ [\mathrm{A}(x) \vee \mathrm{B}(y)] \wedge \forall u\ \mathrm{C}(u)$$
$$\exists y \forall x\ [\mathrm{A}(x) \vee \mathrm{B}(y)] \wedge \exists u\ \mathrm{C}(u)$$
$$\exists y \exists x\ [\mathrm{A}(x) \vee \mathrm{B}(y)] \wedge \forall u\ \mathrm{C}(u)$$
$$\exists y \exists x\ [\mathrm{A}(x) \vee \mathrm{B}(y)] \wedge \exists u\ \mathrm{C}(u)$$

Přeškrtnuté vzory jsou takové, které jsme eliminovali podle pravidla pátého z algoritmu generování kvantifikátorů na začátek vzoru. Přesto jsme ale vygenerovali některé vzory duplicitní, které budou odstraněny během fáze eliminace duplicitních vzorů.

4.3.2.2.4 Různé kombinace

Posledním způsobem generování vzorů jsou kombinace předešlých tří způsobů. V případě jakékoliv kombinace budou platit algoritmy popsané u těchto jednotlivých způsobů. Ukažme si nyní například kombinaci, kdy kvantifikátory budou na začátku vzoru a před prvním predikátem.

Příklad: Ze vzoru A(x, y) ∧ B(x, z) vygenerujeme tyto vzory:

$$\forall x \forall z\ [\forall x \forall y\ \mathrm{A}(x, y) \wedge \mathrm{B}(x, z)]$$
$$\forall x \forall z\ [\forall x \exists y\ \mathrm{A}(x, y) \wedge \mathrm{B}(x, z)]$$
$$\forall x \forall z\ [\exists x \forall y\ \mathrm{A}(x, y) \wedge \mathrm{B}(x, z)]$$
$$\forall x \forall z\ [\exists x \exists y\ \mathrm{A}(x, y) \wedge \mathrm{B}(x, z)]$$
$$\dots$$
$$\exists z \exists x\ [\exists x \exists y\ \mathrm{A}(x, y) \wedge \mathrm{B}(x, z)]$$

U kvantifikátorů, které bezprostředně předcházejí predikátu A, nezáleží dle definice vzoru na pořadí kvantifikování proměnných. U kvantifikátorů na začátku vzoru ovšem na pořadí kvantifikování proměnných záleží, a proto má poslední vzor proměnné u těchto kvantifikátorů, na rozdíl od ostatních vzorů v příkladu, prohozeny. Na pořadí kvantifikování proměnných na začátku vzoru záleží, jelikož jejich pořadí ovlivňuje výsledný řetězec složitosti.

4.3.3 Generování vzorů vyžadujících zavedení existenčního kvantifikátoru

Vzory vyžadující při převádění do Skolemovy klauzulární formy použití kroku zavedení existenčního kvantifikátoru bude systém generovat pro všechny hierarchické řády stejným způsobem. Vstupem do této fáze bude množina vzorů, která je výstupem fáze druhé, tedy generování kvantifikátorů. Výstupem bude množina vzorů, která byla na vstupu, doplněná o vzory, které vzniknou níže uvedeným algoritmem, který systém aplikuje na každý vzor ve vstupní množině.

1. Zkontrolujeme, jestli se na začátku formule nachází existenční kvantifikátor. Jinými slovy zkontrolujeme, zda kořenem syntaktického stromu je vrchol reprezentující existenční kvantifikátor. Pokud ne, pokračujeme dalším vzorem.
2. Pokud ano, potom musíme zkontrolovat, zda se ve vzoru nenachází jiný kvantifikátor takový, že kvantifikuje proměnnou se stejným jménem jako existenční kvantifikátor na začátku formule. Pokud takový kvantifikátor nalezneme, musíme ověřit, zda by po odstranění existenčního kvantifikátoru ze začátku formule nenastala situace, kdy by vzor obsahoval proměnnou se stejným jménem jako vázanou a zároveň volnou, což by znamenalo, že vzor nepatří do jazyka PL1. V takovém případě bychom existenční kvantifikátor ze začátku formule odstranit nemohli a pokračovali bychom dalším vzorem.

   Příklad:
   Mějme vzor $\exists x\ [\mathrm{A}(x) \wedge \forall x \forall y\ \mathrm{B}(x, y)]$. Odebráním existenčního kvantifikátorů ze začátku vzoru získáme $[\mathrm{A}(x) \wedge \forall x \forall y\ \mathrm{B}(x, y)]$, což ale není dobře utvořená formule jazyka PL1, jelikož proměnná x se vyskytuje jako volná a zároveň jako vázaná.

3. Odstraníme existenční kvantifikátor ze začátku formule. Tím získáme nový vzor, který přidáme do množiny výstupních vzorů.

4.3.4 Možnosti dynamického generování až při požadavku uživatele

Jednou z možností, jak efektivně zvýšit počet různých, systémem generovaných vzorů, aniž by se zvětšila velikost datových souborů, je přesouvání negace přes kvantifikátory, pokud je na začátku vzoru negace, bezprostředně následována alespoň dvěma kvantifikátory. Systém ve fázi předgenerování generuje vzory tak, že negaci vkládá na začátek vzoru a její možný přesun neřeší. Algoritmus pro přesunutí negace na začátku vzoru nyní popíšeme.

1. Zvolí-li uživatel generování vzorů, jejichž řetězec obsahuje hodnotu jedna na položce přesun negace přes kvantifikátory, pokračujeme dále. Jinak algoritmus končí.
2. Ze všech vzorů s řetězcem složitosti shodným s řetězcem, který zadal uživatel, vybereme takové, které na začátku mají negaci bezprostředně následovanou alespoň dvěma kvantifikátory.
3. Pro každý vzor vybraný v předešlém kroku a $(n - 1)$-krát (n je počet kvantifikátorů, které bezprostředně následující za negací na začátku vzoru) přesuneme negaci právě přes jeden kvantifikátor doprava, přičemž musíme podle De Morganových pravidel změnit kvantifikátor, přes který negaci přesouváme, a takto vzniklý vzor přidáme do množiny vzorů, které se vrátí uživateli jako výstup. Po poslední iteraci bude mít vzor strukturu takovou, že na jeho začátku bude *n-1* kvantifikátorů, následováno negací a jedním dalším kvantifikátorem.

Příklad: Ze vzoru $\neg\exists x \forall y \exists z\ P(x, y, z)$ tímto algoritmem získáme:

$$\forall x \neg \forall y \exists z\ \mathrm{P}(x, y, z)$$
$$\forall x \exists y \neg \exists z\ \mathrm{P}(x, y, z)$$

Řetězec složitosti takto vygenerovaných vzorů se nezmění, jelikož přes kvantifikátory, přes které negaci přesunujeme, by se stejně negace musela během skolemizace přesunout a jelikož přes polední kvantifikátor nepřesunujeme, tak hodnota položky, odpovídající přesunu negace přes kvantifikátory, zůstane jedna.

Podobným způsobem by bylo možné přesunovat negaci přes kvantifikátory, které se nacházejí před závorkami a přímo před predikáty. U takových přesunů, by ale bylo nutné dávat pozor na strukturu formule, abychom přesunem negace nezměnili řetězec složitosti vygenerovaného vzoru. Tento algoritmus už ale nebude předmětem této práce, myšlenku uvádíme pouze jako další možnost, jak zvýšit počet generovaných vzorů.

4.3.5 Vzory vyžadující eliminaci nadbytečných kvantifikátorů

Další možností, jak zefektivnit fungování systému a redukovat velikost datových souborů je generování vzorů, při jejichž převedení do Skolemovy klauzulární formy je nutné použít krok třetí (eliminace nadbytečných kvantifikátorů), dynamicky až při požadavku uživatele. Tímto zmenšíme velikost datových souborů o ½. Generování těchto vzorů bude probíhat až po algoritmu popsaném v části 4.3.4, následovně:

1. Nalezneme vzory lišící se pouze na položce, odpovídající kroku třetímu - eliminace nadbytečných kvantifikátorů.
2. Do každého nalezeného vzoru přidáme kvantifikátor kvantifikující proměnnou, kterou vzor ještě neobsahuje. Kvantifikátor vložíme v syntaktickém stromu na náhodně zvolené místo *m*, ale takové, že se na cestě od *m* ke kořeni stromu nevyskytuje žádný vrchol odpovídající predikátu. Tímto zajistíme, že se opět bude jednat o formuli jazyka PL1. Zda se bude jednat o všeobecný, nebo existenční kvantifikátor, opět ponecháme na náhodě.

   Příklad: Ze vzoru A chceme získat vzor B, při jehož skolemizaci bude nutné použít krok eliminace nadbytečných kvantifikátorů.

   $$\mathrm{A} = \forall y \exists x \exists z\ [\mathrm{P}(x, z) \wedge \mathrm{Q}(y)]$$
   $$\mathrm{B}_1 = \forall y \exists x \exists q \exists z\ [\mathrm{P}(x, z) \wedge \mathrm{Q}(y)]$$
   $$\mathrm{B}_2 = \forall y \exists x \exists z\ [\mathrm{P}(x, z) \wedge \forall q\ \mathrm{Q}(y)]$$

   V příkladu jsme ukázali dvě možné varianty, jak ze vzoru A získáme vzor B, který se liší pouze ve složce řetězce složitosti, která odpovídá kroku eliminace nadbytečných kvantifikátorů.

Takto jednoduše dynamicky, efektivně, a dokonce s prvky náhody vygenerujeme vzory, při jejichž převedení do Skolemovy klauzulární formy, bude nutné použít krok eliminace nadbytečných kvantifikátorů.

4.3.6 Vzory obsahující funkce a konstanty

Podobně jako v předchozích částech nebudeme do datových souborů generovat vzory obsahující konstanty a funkce, ale takovéto vzory budou generovány dynamicky až při požadavku uživatele. Uživatel bude mít možnost ze zvoleného vzoru nechat systém vytvořit vzor se stejnou strukturou formule a stejným řetězcem složitosti, obsahující konstantu, funkci nebo obojí. Vzory obsahující konstantu se budou generovat takto.

1. U uživatelem vybraného vzoru zvolíme náhodný predikátový symbol *p*.
2. Vygenerujeme náhodná jména pro novou konstantu dle vymezené podmnožiny PL1.
3. Všem výskytům predikátu *p* přidáme na náhodnou pozici konstantu pojmenovanou dle kroku č. 2. Konstantu musíme přidávat do všech výskytů predikátů *p* z důvodu, aby byla zachována stejná arita.

Příklad: Mějme vzor A, ke kterému chceme získat vzor B, obsahující konstantu.

$$A = \forall x \exists z\, [P(x, y) \wedge P(y, y)]$$
$$B = \forall x \exists z\, [P(x, y, a) \wedge P(y, a, y)]$$

Pro přidání funkce do vzoru bychom mohli použít stejný algoritmus jako v případě konstanty. Aby se arita predikátů zbytečně nezvyšovala, algoritmus mírně pozměníme takto.

1. U uživatelem vybraného vzoru zvolíme náhodný predikátový symbol *p*.
2. Vybereme náhodný term *t* v predikátu *p*.
3. Vygenerujeme náhodná jména pro novou funkci dle vymezené podmnožiny PL1.
4. Na pozici termu *t* vložíme novou funkci pojmenovanou dle kroku č. 3, jejímž argumentem se stane term *t*.

Příklad: Mějme vzor A, ke kterému chceme získat vzor B, obsahující funkci.

$$A = \forall x \exists z\, [P(x, y) \wedge P(y, y)]$$
$$B = \forall x \exists z\, [P(x, y) \wedge P(y, f(y)]$$

Bude-li uživatel požadovat, aby vzor obsahoval konstantu i funkci, přidá se nejprve podle výše uvedených algoritmů konstanta a poté funkce, abychom neztratili možnost, že se argumentem funkce stane vložená konstanta.

4.3.7 Generování vzorů nultého hierarchického řádu

V rámci nultého hierarchického řádu máme možnost vygenerovat pouze vzory obsahující jeden predikát a odlišné vzory tak můžeme vytvářet pouze pomocí různých kombinací kvantifikátorů a proměnných u onoho predikátu. Jediným strukturním vzorem nultého hierarchického řádu je tedy vzor.

1. A

Uvažování nultého hierarchického řádu však dává smysl, jelikož takovéto formule jsou přehledné, a proto vhodné především na prvotní seznámení se s algoritmem skolemizace. U formulí nultého hierarchického řádu mohou být během převedení do Skolemovy klauzulární formy použity pouze kroky odpovídající položkám řetězce složitosti.

- Zavedení existenčního uzávěru.
- Eliminace nadbytečných kvantifikátorů.
- Skolemizace konstantou.
- Skolemizace funkcí.

4.3.7.1 Fáze generování proměnných

Generování proměnných u nultého hierarchického řádu bude probíhat dle těchto pravidel.

1. Arita predikátu bude od jedné do maximálně čtyř.
2. Jména proměnných budeme vybírat z pole symbolů [x, y, u, v] a vytvářet tak vzory podle následujících pravidel.
    a. Každý predikát obsahuje alespoň tolik proměnných se jménem na pozici *i* z pole [x, y, u, v] jako se jménem, které je v poli na pozici *i+1*.
    b. Proměnné jsou u predikátu seřazeny dle jmen stejně jako v poli [x, y, u, v].
    c. Každý predikát obsahuje maximálně dvě proměnné se stejným jménem.

    Tímto postupem docílíme, že nebudeme generovat duplicitní vzory.

    Příklad: Pro maximální počet proměnných rovný čtyřem budeme vytvářet tyto vzory:

$$A(x)$$
$$A(x, x)$$
$$A(x, y)$$
$$A(x, x, y)$$
$$A(x, y, u)$$
$$A(x, x, y, y)$$
$$A(x, x, y, u)$$
$$A(x, y, u, v)$$

Parametry arita i maximální počet proměnných se stejným jménem jsou nastavitelné v konfiguračním souboru, mohou se v budoucnu dle potřeby změnit a jejich přesný popis bude uveden společně s dalšími nastavitelnými parametry systému v kapitole 5.

Generování kvantifikátorů bude probíhat pouze způsobem přímo před predikát, popsaným v části 4.3.2.1.2.

Příklad: Ze vzoru A(x, y) vygenerujeme dle kroku třetího tyto vzory:

$$\forall x \forall y \, A(x, y)$$
$$\forall x \exists y \, A(x, y)$$
$$\exists x \forall y \, A(x, y)$$
$$\exists x \exists y \, A(x, y)$$

4.3.8 Generování vzorů prvního hierarchického řádu

U prvního hierarchického řádu máme již větší počet možností, jak generovat vzory. Budeme pracovat se strukturními vzory.

1. $\neg A$
2. A @ B

U vzorů prvního hierarchického řádu mohou být během převedení do Skolemovy klauzulární formy použity pouze kroky odpovídající těmto položkám řetězce.

- Zavedení existenčního uzávěru.
- Eliminace nadbytečných kvantifikátorů.
- Přejmenování proměnných.
- Eliminace implikace.
- Eliminace ekvivalence.
- Přesunutí negace přes kvantifikátory.
- Lze přesunout alespoň jeden kvantifikátor a zároveň všechny.
- Lze přesunout alespoň jeden kvantifikátor, ale ne všechny.
- Skolemizace konstantou.
- Skolemizace funkcí.
- Přesunutí kvantifikátorů doleva.

Pro strukturní vzor $\neg A$ budeme generovat vzory následujícím postupem:

1. Vygenerujeme vzory nultého hierarchického řádu ze vzoru A.
2. Na začátek každého vzoru vygenerovaného v kroku jedna vložíme negaci.

Z hlediska řetězce složitosti nemá smysl vkládat negaci nikam jinam, než na začátky vzorů, které jsou generovány v kroku prvním. Pokud bychom negaci vkládali kdekoliv mezi kvantifikátory, docházelo by k navýšení už tak velkého počtu vzorů, proto budou takovéto vzory generovány dynamicky, jak je popsáno v části 4.3.6. V případě, že by negace byla až za kvantifikátory, byl by řetězec složitosti převedení do klauzulární Skolemovy formy totožný jako za situace, ve které by nebyla do vzoru vložena žádná negace. Ale takovýto vzor byl zajisté přidán do databáze již během generování vzorů nultého hierarchického řádu.

Příklad:

$$\forall x \neg \mathrm{P}(x)$$

Tento vzor má evidentně stejný řetězec složitosti jako vzor $\forall x\ \mathrm{P}(x)$, který jsme už ale přidali do databáze vzorů během generování nultého hierarchického řádu.

Pro ostatní strukturní vzory budeme generování realizovat standardně třemi fázemi, jak bylo popsáno v části 4.3.2.

4.3.8.1 Fáze generování proměnných

Proměnné přidáváme k predikátům A, B strukturního vzoru v kombinacích uvedených v tabulce č. 3. (V první závorce jsou proměnné predikátu A, ve druhé proměnné predikátu B).

Tabulka 3: Proměnné vkládané do vzorů prvního hierarchického řádu

Kombinace vkládaných proměnných	Vzory vzniklé vkládáním proměnných do strukturního vzoru: $\mathrm{A} \wedge \mathrm{B}$
(x) (x)	$\mathrm{A}(x) \wedge \mathrm{B}(x)$
(x) (y)	$\mathrm{A}(x) \wedge \mathrm{B}(y)$
(x, y) (x)	$\mathrm{A}(x, y) \wedge \mathrm{B}(x)$
(x, y) (u)	$\mathrm{A}(x, y) \wedge \mathrm{B}(y)$
(x, y) (x, y)	$\mathrm{A}(x, y) \wedge \mathrm{B}(x, y)$
(x, y) (x, u)	$\mathrm{A}(x, y) \wedge \mathrm{B}(x, u)$
(x, y) (u, v)	$\mathrm{A}(x, y) \wedge \mathrm{B}(u, v)$
(x, y, u) (x)	$\mathrm{A}(x, y, u) \wedge \mathrm{B}(x)$
(x, y, u) (v)	$\mathrm{A}(x, y, u) \wedge \mathrm{B}(v)$
(x, y, u) (x, y)	$\mathrm{A}(x, y, u) \wedge \mathrm{B}(x, y)$
(x, y, u) (x, v)	$\mathrm{A}(x, y, u) \wedge \mathrm{B}(x, v)$
(x, y, u) (v, z)	$\mathrm{A}(x, y, u) \wedge \mathrm{B}(v, z)$

Kombinace vkládaných proměnných jsou opět nastavitelným parametrem systému.

Výjimkou při generování proměnných je vzor A ⊃ B, u kterého z důvodu, že implikace není komutativní spojka, nemůžeme pomocí substitučních pravidel prohodit predikáty A, B a získat tak některé příklady, které by jinak vznikly právě prohozením. Proto v případě vzoru A ⊃ B ještě přidáme takové kombinace k výše uvedeným, které z nich vzniknou prohozením části v závorkách u kombinací, kde se počet proměnných v závorkách liší. Pro vzor A ⊃ B tak navíc vzniknou kombinace uvedené v tabulce č. 4.

Tabulka 4: Proměnné vkládané navíc do vzoru A ⊃ B

Kombinace navíc vkládaných proměnných	Vzory navíc vzniklé z A ⊃ B
(x) (x, y)	$A(x) \supset B(x, y)$
(u) (x, y)	$A(u) \supset B(x, y)$
(x) (x, y, u)	$A(x) \supset B(x, y, u)$
(v) (x, y, u)	$A(v) \supset B(x, y, u)$
(x, y) (x, y, u)	$A(x, y) \supset B(x, y, u)$
(x, v) (x, y, u)	$A(x, v) \supset B(x, y, u)$
(v, z) (x, y, u)	$A(v, z) \supset B(x, y, u)$

4.3.8.2 Fáze generování kvantifikátorů

Pro každý vzor, vniklý ve fázi generování proměnných, budeme kvantifikátory generovat těmito dříve popsanými způsoby:

1. Kvantifikátory budou před jednotlivými predikáty vzoru.
2. Kvantifikátory budou na začátku vzoru.
3. Kvantifikátory budou na začátku vzoru a před prvním predikátem.
4. Kvantifikátory budou na začátku vzoru a před druhým predikátem.

Posledním způsobem, který dříve popsaný nebyl, bude takový, že kvantifikátory budou na začátku vzoru a kvantifikátory kvantifikující proměnné, které se nevyskytují v druhém predikátu, tedy B, přesuneme před predikát A. Pokud je počet takových kvantifikátorů nula, nebo je roven počtu různých proměnných u predikátu A, pak pro daný vzor tímto způsobem kvantifikátory generovat nebudeme, jelikož výsledné vzory by byly stejné jako vzory získané předešlými způsoby.

*Příklad: Ze vzoru A(*x, y*) ∧ B(*x, u*) vygenerujeme tímto způsobem tyto vzory:*

$$\forall x \forall u\, [\forall y A(x, y) \wedge B(x, u)]$$
$$\forall x \forall u\, [\exists y A(x, y) \wedge B(x, u)]$$
$$\forall x \exists u\, [\forall y A(x, y) \wedge B(x, u)]$$
$$\forall x \exists u\, [\exists y A(x, y) \wedge B(x, u)]$$
$$\exists x \forall u\, [\forall y A(x, y) \wedge B(x, u)]$$

$$\exists x \forall u\, [\exists y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\exists x \exists u\, [\forall y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\exists x \exists u\, [\exists y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\forall u \forall x\, [\forall y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\forall u \forall x\, [\exists y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\forall u \exists x\, [\forall y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\forall u \exists x\, [\exists y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\exists u \forall x\, [\forall y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\exists u \forall x\, [\exists y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\exists u \exists x\, [\forall y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$
$$\exists u \exists x\, [\exists y \mathrm{A}(x, y) \wedge \mathrm{B}(x, u)]$$

V případě vzoru A $\supset$ B provedeme vygenerování způsobem takovým, že analogicky přesuneme kvantifikátory, které kvantifikují proměnné, které se nevyskytují u predikátu A, ze začátku formule před predikát B.

4.3.9 Generování vzorů druhého hierarchického řádu

U druhého hierarchického řádu je již počet možností, jak generovat vzory, velký. Strukturní vzory jsou nastavitelným parametrem systému a v případě potřeby je bude možné v budoucnu bez zásahu do implementace změnit. V této práci použijeme pro generování vzorů druhého hierarchického řádu strukturní vzory uvedené v tabulce č. 5.

Tabulka 5: Strukturní vzory druhého hierarchického řádu

¬A @ ¬B	(A @ B) $ C
¬(A @ B)	(A @ B) $ ¬C
¬A $\supset$ B	(A $ B) ≡ C
A $ (B @ C)	(A $ B) ≡ ¬C
¬A $ (B @ C)	(A @ B) $ (C @ D)
A ≡ (B $ C)	(A $ B) ≡ (C $ D)
¬A ≡ (B $ C)	

Pro strukturní vzory o dvou predikátech, tedy (¬A @ ¬B), ¬(A @ B) a (¬A $\supset$ B) bude generování probíhat tak, že vygenerujeme vzory prvního hierarchického řádu, ale pouze ze strukturního vzoru A @ B. Tyto vzory budou jen mezivýsledek při generování. Pro každý takto vygenerovaný vzor *v* budeme dále postupovat následovně a vytvářet tak množinu *V*, která bude výstupem algoritmu:

1. Pro strukturní vzor (¬A @ ¬B) vložíme negaci před predikáty A, B, a pak ji přesuneme doleva přes všechny kvantifikátory, které jsou bezprostředně před predikáty A, B. Tímto vznikne nový vzor, který přidáme do množiny *V*.
2. Pro strukturní vzor ¬(A @ B) vložíme negaci na úplný začátek vzoru *v*. Nově vzniklý vzor přidáme do množiny *V*. Je-li vzor *v* v prenexním tvaru, nebo obsahuje-li implikaci a kvantifikátory jsou pouze na začátku vzoru nebo před predikátem A, pak navíc vložíme negaci ještě bezprostředně před jedinou binární logickou spojku. Takto vzniklý vzor přidáme do množiny *V*, protože bude mít na rozdíl od vzoru, který má negaci na samém začátku, jiný řetězec složitosti. Konkrétně se bude lišit na položce odpovídající přesunu negace přes kvantifikátory, na které bude mít hodnotu nula.
3. Je-li binární logickou spojkou ve vzoru *v* implikace, pak vložíme negaci před predikát A, a přesuneme ji doleva přes všechny bezprostředně předcházející kvantifikátory a vzniklý vzor přidáme do množiny *V*.

4.3.9.1 Fáze generování termů

Pro vzory o třech a čtyřech predikátech budeme generování realizovat standardně třemi fázemi, jak bylo popsáno v části 4.3.2 fáze generování termů. Pouze u vzorů obsahujících negaci přesuneme ještě negaci doleva přes kvantifikátory, které jí bezprostředně předcházejí. Proměnné budeme predikátům přidávat podle kombinací v tabulkách č. 6 a 7.

Tabulka 6: Proměnné vkládané do vzorů druhého hierarchického řádu se třemi predikáty

Kombinace vkládaných proměnných	Vzory vzniklé vkládáním proměnných do strukturního vzoru: $A \equiv (B \wedge C)$
(x) (x) (x)	$A(x) \equiv (B(x) \wedge C(x))$
(x) (y) (x)	$A(x) \equiv (B(y) \wedge C(x))$
(x) (y) (y)	$A(x) \equiv (B(y) \wedge C(y))$
(x) (y) (u)	$A(x) \equiv (B(y) \wedge C(u))$
(x, y) (x) (x)	$A(x, y) \equiv (B(x) \wedge C(x))$
(x, y) (x) (u)	$A(x, y) \equiv (B(x) \wedge C(u))$
(x, y) (u) (x)	$A(x, y) \equiv (B(u) \wedge C(x))$
(x, y) (u) (v)	$A(x, y) \equiv (B(u) \wedge C(v))$
(x) (x) (x, y)	$A(x) \equiv (B(x) \wedge C(x, y))$
(x) (u) (x, y)	$A(x) \equiv (B(u) \wedge C(x, y))$
(u) (x) (x, y)	$A(u) \equiv (B(x) \wedge C(x, y))$
(u) (v) (x, y)	$A(u) \equiv (B(v) \wedge C(x, y))$
(x, y) (x) (x, y)	$A(x, y) \equiv (B(x) \wedge C(x, y))$
(x, y) (x) (x, u)	$A(x, y) \equiv (B(x) \wedge C(x, u))$
(x, y) (x) (u, v)	$A(x, y) \equiv (B(x) \wedge C(u, v))$
(x, y) (u) (x, y)	$A(x, y) \equiv (B(u) \wedge C(x, y))$

(x, y) (u) (x, v)	$A(x, y) \equiv (B(u) \wedge C(x, v))$
(x, y) (u) (v, w)	$A(x, y) \equiv (B(u) \wedge C(v, w))$

Všimněme si, že některé vzory se vygenerují duplicitní, to ale nevadí, jelikož duplicity budou odstraněny během fáze eliminace duplicitních vzorů.

Tabulka 7: Proměnné vkládané do vzorů druhého hierarchického řádu se čtyřmi predikáty

Kombinace vkládaných proměnných	Vzory vzniklé vkládáním proměnných do strukturního vzoru: $A \equiv (B \wedge C)$
(x) (x) (x) (x)	$A(x) \equiv (B(x) \wedge C(x))$
(x) (y) (x) (y)	$A(x) \equiv (B(y) \wedge C(x))$
(x) (x) (y) (y)	$A(x) \equiv (B(y) \wedge C(y))$
(x) (x) (y) (u)	$A(x) \equiv (B(y) \wedge C(u))$
(x) (y) (x) (u)	$A(x, y) \equiv (B(x) \wedge C(x))$
(x) (y) (u) (v)	$A(x, y) \equiv (B(x) \wedge C(u))$
(x, y) (y) (u) (v)	$A(x, y) \equiv (B(u) \wedge C(x))$
(x, y) (u) (v) (w)	$A(x, y) \equiv (B(u) \wedge C(v))$

4.3.9.2 Fáze generování kvantifikátorů

Kvantifikátory budeme pro vzory o třech a čtyřech literálech generovat následujícími způsoby:

1. Kvantifikátory budou před jednotlivými predikáty vzoru.
2. Kvantifikátory budou na začátku vzoru.

Pro vzory se vzniklé ze strukturních vzorů A @ (B @ C) a ¬A @ (B @ C) budeme navíc kvantifikátory generovat následujícími způsoby:

1. Kvantifikátory budou před výrazem (B @ C) a na začátku vzoru.
2. Kvantifikátory budou před výrazem (B @ C) a před predikátem A.
3. Kvantifikátory budou před predikátem B a na začátku vzoru.
4. Kvantifikátory budou před predikátem C a na začátku vzoru.
5. Kvantifikátory budou před predikáty A, B a na začátku vzoru.
6. Kvantifikátory budou před predikáty A, C a na začátku vzoru.
7. Kvantifikátory budou před predikáty B, C a na začátku vzoru.

Pro vzory, vzniklé ze strukturních vzorů (A @ B) @ C a (A @ B) @ ¬C, budeme navíc kvantifikátory generovat následujícími způsoby:

1. Kvantifikátory budou před výrazem (A @ B) a na začátku vzoru.
2. Kvantifikátory budou před výrazem (A @ B) a před predikátem C.

3. Kvantifikátory budou před predikátem A a na začátku vzoru.
4. Kvantifikátory budou před predikátem B a na začátku vzoru.
5. Kvantifikátory budou před predikáty A, B a na začátku vzoru.
6. Kvantifikátory budou před predikáty A, C a na začátku vzoru.
7. Kvantifikátory budou před predikáty B, C a na začátku vzoru.

Pro vzory, vzniklé ze strukturních vzorů (A @ B) @ (C @ D), budeme navíc kvantifikátory generovat následujícími způsoby:

1. Kvantifikátory budou před výrazem (A @ B) a na začátku vzoru.
2. Kvantifikátory budou před výrazem (C @ D) a na začátku vzoru.
3. Kvantifikátory budou před výrazy (A @ B), (C @ D).
4. Kvantifikátory budou před predikáty A, B a na začátku vzoru.
5. Kvantifikátory budou před predikáty C, D a na začátku vzoru.
6. Kvantifikátory budou před predikáty A, C a na začátku vzoru.
7. Kvantifikátory budou před predikáty B, D a na začátku vzoru.
8. Kvantifikátory budou před výrazem (A @ B) a před predikáty C, D.
9. Kvantifikátory budou před výrazem (C @ D) a před predikáty A, B.

4.3.10 Generování vzorů třetího hierarchického řádu

Posledním generovaným hierarchickým řádem je řád třetí. Počet možností, jak generovat vzory, je u tohoto řádu natolik velký, že se z praktických důvodů musíme omezit jen na několik málo z nich. Strukturní vzory budou opět nastavitelným parametrem systému a v případě potřeby je bude možné v budoucnu bez zásahu do implementace změnit. Systém bude schopen pracovat se strukturními vzory o třech, čtyřech a šesti predikátech V této práci budeme pro generování vzorů třetího hierarchického řádu používat strukturní vzory, uvedené v tabulce č. 8.

Tabulka 8: Strukturní vzory třetího hierarchického řádu

¬[A $ (B @ C)]	(A ⊃ ¬B) @ ¬(C $ D)
¬[¬A $ (B @ C)]	¬(A $ B) $ ¬(C $ D)
¬[A ≡ (B $ C)]	¬[(A $ B) $ (C $ D)]
¬[¬A ≡ (B $ C)]	(A ∨ B) ∧ [(C @ D) * (E $ F)]
¬[(A @ B) $ C]	(A ∨ B) ∧ [(C $ D) * (E ≡ F)]
¬[(A @ B) $ ¬C]	(A ∨ B) ∧ [(C $ D) ⊃ (E $ F)]
¬[(A $ B) ≡ C]	(A ∨ B) ∧ [(C ∧ D) ⊃ (E ≡ F)]
¬[(A $ B) ≡ ¬C)]	(¬A ∨ ¬B) ∧ [(C @ D) * (E $ F)]
A $ ¬(B @ C)	(¬A ∨ ¬B) ∧ [(C $ D) * (E ≡ F)]
¬A $ ¬(B @ C)	(¬A ∨ ¬B) ∧ [(C $ D) ⊃ (E $ F)]
A ≡ ¬(B $ C)	(¬A ∨ ¬B) ∧ [(C ∧ D) ⊃ (E ≡ F)]
¬A ≡ ¬(B $ C)	(A ⊃ ¬B) ∧ [(C @ D) * (E $ F)]

¬(A @ B) $ C	(A ⊃ ¬B) ∧ [(C $ D) * (E ≡ F)]
¬(A @ B) $ ¬C	(A ⊃ ¬B) ∧ [(C $ D) ⊃ (E $ F)]
¬(A $ B) ≡ C	(A ⊃ ¬B) ∧ [(C ∧ D) ⊃ (E ≡ F)]
¬(A $ B) ≡ ¬C)	(¬A ⊃ B) ∧ [(C @ D) * (E $ F)]
¬A $ (¬B @ ¬C)	(¬A ⊃ B) ∧ [(C $ D) * (E ≡ F)]
A ⊃ (¬B @ ¬C)	(¬A ⊃ B) ∧ [(C $ D) ⊃ (E $ F)]
¬A ≡ (¬B $ ¬C)	(¬A ⊃ B) ∧ [(C ∧ D) ⊃ (E ≡ F)]
(¬A @ ¬B) $ ¬C	¬(A ∧ B) ∧ [(C @ D) * (E $ F)]
(¬A $ ¬B) ≡ ¬C	¬(A ∧ B) ∧ [(C $ D) * (E ≡ F)]
(¬A ⊃ B) $ (C + D)	¬(A ∧ B) ∧ [(C $ D) ⊃ (E $ F)]
(¬A ⊃ B) ≡ (C * D)	¬(A ∧ B) ∧ [(C ∧ D) ⊃ (E ≡ F)]
(¬A ⊃ B) @ (¬C ⊃ D)	[(A @ B) * (C $ D)] ∧ (E ∨ F)
(A + B) $ (¬C ⊃ D)	[(A $ B) * (C ≡ D)] ∧ (E ∨ F)
(A * B) ≡ (¬C ⊃ D)	[(A $ B) ⊃ (C $ D)] ∧ (E ∨ F)
(¬A + ¬B) $ (¬C + ¬D)	[(A ∧ B) ⊃ (C ≡ D)] ∧ (E ∨ F)
(¬A * ¬B) ≡ (¬C * ¬D)	[(A @ B) * (C $ D)] ∧ (¬E ∨ ¬F)
(A ⊃ ¬B) $ (¬C + ¬D)	[(A $ B) * (C ≡ D)] ∧ (¬E ∨ ¬F)
(A ⊃ ¬B) ≡ (¬C * ¬D)	[(A $ B) ⊃ (C $ D)] ∧ (¬E ∨ ¬F)
(A ⊃ ¬B) @ (C ⊃ ¬D)	[(A ∧ B) ⊃ (C ≡ D)] ∧ (¬E ∨ ¬F)
(¬A + ¬B) $ (C ⊃ ¬D)	[(A @ B) * (C $ D)] ∧ (E ⊃ ¬F)
(¬A * ¬B) ≡ (C ⊃ ¬D)	[(A $ B) * (C ≡ D)] ∧ (E ⊃ ¬F)
¬(A $ B) $ (C @ D)	[(A $ B) ⊃ (C $ D)] ∧ (E ⊃ ¬F)
¬(A $ B) ≡ (C * D)	[(A ∧ B) ⊃ (C ≡ D)] ∧ (E ⊃ ¬F)
¬(A $ B) @ (¬C ⊃ D)	[(A @ B) * (C $ D)] ∧ (¬E ⊃ F)
(A @ B) $ ¬(C @ D)	[(A $ B) * (C ≡ D)] ∧ (¬E ⊃ F)
(A * B) ≡ ¬(C $ D)	[(A $ B) ⊃ (C $ D)] ∧ (¬E ⊃ F)
(¬A ⊃ B) @ ¬(C $ D)	[(A ∧ B) ⊃ (C ≡ D)] ∧ (¬E ⊃ F)
¬(A $ B) $ (¬C @ ¬D)	[(A @ B) * (C $ D)] ∧ ¬(E ∧ F)
¬(A $ B) ≡ (¬C * ¬D)	[(A $ B) * (C ≡ D)] ∧ ¬(E ∧ F)
¬(A $ B) @ (C ⊃ ¬D)	[(A $ B) ⊃ (C $ D)] ∧ ¬(E ∧ F)
(¬A @ ¬B) $ ¬(C @ D)	[(A ∧ B) ⊃ (C ≡ D)] ∧ ¬(E ∧ F)
(¬A * ¬B) ≡ ¬(C $ D)	

Pro každý strukturní vzor *v* o třech nebo čtyřech predikátech budeme generování realizovat následovně:

1. Ze strukturního vzoru *v* vytvoříme odebráním všech negací strukturní vzor *v′*, jehož hierarchický řád je dva. Počet negací, které se ve vzoru *v* nacházejí, označíme *n*.
2. Pro strukturní vzor *v′* vygenerujeme dle algoritmu pro generování vzorů druhého hierarchického řádu, který je popsaný v části 7.3.9, množinu vzorů *V*, která se nakonec stane výstupem algoritmu.
3. Nyní pro *i=1* až *n* zopakujeme.
4. Pro každý vzor *u* z množiny *V* vložíme zpět do vzoru *u* *i*-tou negaci, kterou jsme odebrali v kroku prvním. Negaci posuneme doleva přes všechny kvantifikátory, které ji bezprostředně předchází. Pokud navíc není po vložení negace do vzoru *u* jejím přímým následovníkem predikát a podstrom negace neobsahuje žádný kvantifikátor, přidáme do množiny *V* navíc vzor, kde jsme negaci nepřesunuli.

4.3.10.1 Fáze generování termů

Pro vzory o šesti predikátech budeme generování realizovat standardně třemi fázemi, jak bylo popsáno v části 4.3.2 fáze generování termů. Pouze u vzorů obsahujících negaci bude nakonec negace přesunuta doleva přes kvantifikátory, které ji bezprostředně předcházejí. V případě, že následníkem negace ve strukturním vzoru není predikát, ale binární spojka a zároveň výraz, který za negací následuje, neobsahuje žádný kvantifikátor, pak ještě navíc vytvoříme vzor, ve kterém negace přes kvantifikátory přesunuta nebude. Proměnné budeme k predikátům přidávat podle kombinací v tabulce č. 9.

Tabulka 9: Proměnné vkládané do vzorů třetího hierarchického řádu se šesti predikáty

Kombinace vkládaných proměnných	Vzory vzniklé vkládáním proměnných do strukturního vzoru: $(A \vee B) \wedge [(C \vee D) \wedge (E \vee F)]$
(x) (x) (y) (y) (u) (u)	$(A(x) \vee B(x)) \wedge [(C(y) \vee D(y)) \wedge (E(u) \vee F(u))]$
(x) (y) (x) (y) (u) (v)	$(A(x) \vee B(y)) \wedge [(C(x) \vee D(y)) \wedge (E(u) \vee F(v))]$
(u) (v) (x) (y) (x) (y)	$(A(u) \vee B(v)) \wedge [(C(x) \vee D(y)) \wedge (E(x) \vee F(y))]$
(x) (y) (u) (v) (w) (z)	$(A(x) \vee B(y)) \wedge [(C(u) \vee D(v)) \wedge (E(w) \vee F(z))]$

4.3.10.2 Fáze generování kvantifikátorů

Kvantifikátory budeme pro vzory o šesti literálech generovat následujícími způsoby:

1. Kvantifikátory budou před jednotlivými predikáty vzoru.
2. Kvantifikátory budou na začátku vzoru.
3. Kvantifikátory budou před výrazy (A @ B), (C @ D) a (E @ F).
4. Kvantifikátory budou před výrazy (A @ B), (C @ D) a na začátku vzoru.
5. Kvantifikátory budou před výrazy (A @ B), (E @ F) a na začátku vzoru.

6. Kvantifikátory budou před výrazy (C @ D), (E @ F) a na začátku vzoru.
7. Kvantifikátory budou před výrazy (A @ B), (C @ D) a před predikáty E a F.
8. Kvantifikátory budou před výrazy (A @ B), (E @ F) a před predikáty C a D.
9. Kvantifikátory budou před výrazy (C @ D), (E @ F) a před predikáty A a B.

Pro vzory, vzniklé ze strukturních vzorů (A @ B) $\wedge$ [(C @ D) @ (E @ F)] (u strukturních vzorů zanedbáváme negace), budeme navíc kvantifikátory generovat následujícími způsoby:

1. Kvantifikátory budou před výrazy (A @ B) a [(C @ D) @ (E @ F)].
2. Kvantifikátory budou před výrazem (A @ B) na začátku vzoru.
3. Kvantifikátory budou před výrazem [(C @ D) @ (E @ F)] a na začátku vzoru.
4. Kvantifikátory budou před výrazem (A @ B) a před predikáty C, D, E a F.
5. Kvantifikátory budou před výrazem [(C @ D) @ (E @ F)] a před predikáty A a B.

Pro vzory, vzniklé ze strukturních vzorů [(A @ B) @ (C @ D)] $\wedge$ (E @ F) (u strukturních vzorů zanedbáváme negace), budeme navíc kvantifikátory generovat následujícími způsoby:

1. Kvantifikátory budou před výrazy [(A @ B) @ (C @ D)] a (E @ F).
2. Kvantifikátory budou před výrazem [(A @ B) @ (C @ D)] a na začátku vzoru.
3. Kvantifikátory budou před výrazem (E @ F) na začátku vzoru.
4. Kvantifikátory budou před výrazem [(A @ B) @ (C @ D)] a před predikáty E a F.
5. Kvantifikátory budou před výrazem (E @ F) a před predikáty A, B, C a D.

4.3.11 Eliminace duplicitních vzorů

Během generování vzorů jednotlivých hierarchických řádů mohou vznikat duplicitní vzory. Tyto duplicity je nutné eliminovat, protože by zbytečně zvětšovaly velikost datových souborů, a také by mohly způsobovat zmatení uživatelů systému. Eliminovat vzory začneme až v momentě, kdy budeme mít vypočteny řetězce složitosti, jelikož každé dva duplicitní vzory musí mít nutně stejný řetězec složitosti. Hledat duplicity proto budeme vždy jen mezi vzory se stejným řetězcem složitosti, což znatelně zvýší efektivitu.

Abychom mohli duplicitní vzory detekovat, musíme umět vzory převést do formy, ve kterém je budeme porovnávat. Tuto formu získáme úpravami vzoru ve dvou krocích:

1. Úpravy struktury vzoru.
2. Přeskupení termů.

4.3.11.1 Úpravy struktury vzoru

Během úpravy struktury vzoru budeme používat pouze asociativní a komutativní zákony, které budeme aplikovat vždy pouze na výrazy takové, že v rámci jednoho výrazu jsou všechny binární logické spojky stejné a jsou zároveň komutativní a asociativní. Připomínáme, že ve vymezené podmnožině jazyka PL1 se jedná o spojky $\wedge$, $\vee$ a $\equiv$.

Nejprve musíme definovat, jak budeme porovnávat jednotlivé vrcholy syntaktického stromu. Na množině typů syntaktických vrcholů zvolíme následující uspořádání. Pořadí si můžeme libovolně měnit, pouze jej pak musíme dodržovat během porovnávání. Typy vrcholů jsou seřazeny sestupně.

1. Všeobecný kvantifikátor
2. Existenční kvantifikátor
3. Predikát
4. Ekvivalence
5. Implikace
6. Konjunkce
7. Disjunkce
8. Negace
9. Funkce
10. Konstanta
11. Předmětová proměnná

Nyní uvedeme, jak porovnat dva výrazy A, B. Porovnávat budeme jejich syntaktické stromy. Definujeme, že výraz A > B pokud:

1. Počet vrcholů reprezentujících predikát v syntaktickém stromu A je větší než počet vrcholů reprezentujících predikát v syntaktickém stromu B. Pokud je menší, pak A < B. Pokud stejný, pokračujeme bodem 2.
2. Počet vrcholů syntaktického stromu A je větší než počet vrcholů syntaktického stromu B. Pokud je menší, pak A < B. Pokud je stejný, pokračujeme bodem 3.
3. Průchodem do šířky syntaktického stromu A získáme pole vrcholů p_A. Obdobně získáme pole p_B. Nyní pro i = 1 až počet prvků p_A porovnáme i-tou položku pole p_A s i-tou položkou pole p_B. Pokud je $p_A[i] > p_B[i]$, pak A > B. Pokud je $p_A[i] < p_B[i]$, pak A < B. Jinak pokračujeme další položkou. Pokud i = počet prvků p_A, pak A = B. Tady si musíme uvědomit, že nejde o rovnost z hlediska sémantiky výrazu, ale pouze z hlediska porovnání pro účely převedení vzoru do formy, ve které jej lze porovnávat.

Strukturu vzoru budeme upravovat podle pravidel v tomto pořadí:

1. Máme-li výraz (A + (B + C)), pak uplatníme asociativní zákon tak, abychom získali výraz ((A + B) + C), předpokládáme, že symbol + zastupuje ve výrazu vždy stejnou binární spojku.
2. Máme-li výraz (A + B) a pro výrazy A, B platí A < B, pak uplatníme komutativní zákon na spojku + tak, abychom získali výraz (B + A).
3. Máme-li výraz ((A + B) + C) a pro výrazy A, B, C platí A > B a B < C, pak uplatníme asociativní a komutativní zákony tak, abychom získali výraz ((A + C) + B). Opět předpokládáme, že symbol + zastupuje ve výrazu vždy stejnou binární spojku.

4.3.11.2 Přeskupení termů

Ve vzoru, jehož strukturu jsme upravili, nyní musíme přeskupit termy v jednotlivých predikátech a funkcích tak, aby pro každý predikát a funkci platilo:

1. Je-li p n-ární predikát s termy $t_1...t_n$, pak pro všechna $i \in \{1, \ldots, n\text{-}1\}$ je $t_i \geq t_{i+1}$.
2. Je-li f n-ární funkce s termy $t_1...t_n$, pak pro všechna $i \in \{1, \ldots, n\text{-}1\}$ je $t_i \geq t_{i+1}$.

Termy t_i, t_{i+1} porovnáváme pomocí algoritmu pro porovnání dvou výrazů uvedeného v kapitole 4.3.11.1. Na pořadí termů podle definice vzoru nezáleží, takže touto úpravou vzor nezměníme.

Nyní již umíme vzory převést do formy, ve které je lze mezi sebou navzájem porovnat. Ze vzorů se stejným řetězcem složitosti budeme konstruovat množinu vzorů V, ve které již nebudou žádné duplicitní vzory. Do množiny V přidáme jeden vzor a pro všechny ostatní vzory v se stejným řetězcem složitosti budeme postupovat následovně:

1. Porovnáme v postupně se všemi vzory z množiny V na počet predikátů a vrcholů syntaktického stromu. Pokud ani jednou nedojde ke shodě, tak přidáme v do V a pokračujeme dalším vzorem, jinak pokračujeme bodem 2.
2. Porovnáme syntaktický strom formy vzoru v, která vznikla dle algoritmů popsaných v kapitolách 4.3.11.1 a 4.3.11.2 se syntaktickými stromy forem vzorů z množiny V, které vznikly podle stejného algoritmu. K porovnání syntaktických stromu použijeme algoritmus pro porovnání dvou výrazů popsaný v kapitole 4.3.11.1. Dojde-li alespoň jednou ke shodě, pak je v duplicitní vzor, který do množiny V nepřidáme. V opačném případě vzor v do množiny V přidáme.

Krok první není nezbytně potřeba provádět. V práci jej uvádíme zejména z důvodu zvýšení efektivity.

4.3.12 Substituce do vzorů a vytváření příkladů

Systém je vyvíjen pro účely výuky. Uplatnění nalezne zejména při tvorbě příkladů na procvičování algoritmu skolemizace, a také pro tvorbu písemkových a zkouškových příkladů. Jelikož se v dnešní době připravuje více verzí písemek, je žádoucí, aby systém dokázal vygenerovat různá zadání se stejnou složitostí. Skolemizace různých příkladů se stejným řetězcem složitosti však nemusí být pro studenty stejně obtížná. Z tohoto důvodu definujeme postupy, jak z některých vzorů vytvářet více různých příkladů, jejichž obtížnost řešení bude naprosto stejná. Přesto se však budou na první pohled jevit studentům jako různé příklady.

Pokusíme se nyní definovat, jak z konkrétního vzoru vytvářet takovéto příklady. Způsob vytváření příkladů bude vycházet z definice pojmu vzor, který jsme uvedli v kapitole 4.2. Způsoby vytváření příkladů rozdělíme do dvou samostatných kategorií.

1. Přejmenovávání predikátů, funkcí a proměnných.
2. Změny struktury vzoru.

Změnu jmen můžeme realizovat na jménech predikátů, funkcí i proměnných. V navrhovaném systému se omezíme pouze na změny jmen predikátů. Toto přejmenování predikátů bude pro jeden konkrétní vzor probíhat následovně:

1. Vygenerujeme všechny permutace unikátních jmen predikátů, které se vyskytují v daném vzoru.
2. Provedeme přejmenování podle každé permutace. Musíme si však dát pozor na správné přejmenování, zejména v případě, že vzor obsahuje dva a více predikátů se stejným jménem. Takové predikáty musí mít po přejmenování opět stejné jméno.

   Příklad: Mějme vzor $\forall y \{[A(a) \wedge \forall x\ B(x, y)] \supset [A(y) \wedge C(a)]$, u kterého budeme přejmenovávat predikáty dle pravidel výše:

$$\forall y \{[A(a) \wedge \forall x\ B(x, y)] \supset [A(y) \wedge C(a)]$$
$$\forall y \{[A(a) \wedge \forall x\ C(x, y)] \supset [A(y) \wedge B(a)]$$
$$\forall y \{[B(a) \wedge \forall x\ A(x, y)] \supset [B(y) \wedge C(a)]$$
$$\forall y \{[B(a) \wedge \forall x\ C(x, y)] \supset [B(y) \wedge A(a)]$$
$$\forall y \{[C(a) \wedge \forall x\ A(x, y)] \supset [C(y) \wedge B(a)]$$
$$\forall y \{[C(a) \wedge \forall x\ B(x, y)] \supset [C(y) \wedge A(a)]$$

Změny ve struktuře formulí budou složitější na implementaci. Mají ovšem výhodu v tom, že lépe studentům skrývají skutečnost, že více různých příkladů je pořád tentýž stejný vzor. Tím se stávají vhodnějšími pro tvorbu zadání písemek a zkoušek. Změnu struktury vzoru budeme realizovat takto:

1. Ve vzoru identifikujeme všechny výrazy takové, že v rámci jednoho výrazu jsou všechny binární logické spojky stejné a jsou zároveň komutativní a asociativní. Připomínáme, že ve vymezené podmnožině jazyka PL1 se jedná o spojky $\wedge$, $\vee$ a $\equiv$.
2. Pro každý identifikovaný výraz z bodu jedna uplatníme asociativní a komutativní zákon, a všechny jejich kombinace na všechny možné podmnožiny spojek tohoto výrazu. Tímto získáme příklady lišící se svou strukturou.

Výše uvedenými úpravami struktury vzoru mohou vznikat duplicitní příklady. Duplicitní příklady rozpoznáme pomocí algoritmu pro porovnání dvou výrazů, který je uveden v kapitole 4.3.11.1. Následně tyto nalezené duplicitní příklady eliminujeme.

Příklad: Mějme vzor [A(a) ∧ B(b)] ∧ ¬C(a), u kterého budeme měnit strukturu dle pravidel uvedených výše:

$$[A(a) \wedge B(a, b)] \wedge \neg C(a)$$
$$[B(a, b) \wedge A(a)] \wedge \neg C(a)$$
$$\neg C(a) \wedge [A(a) \wedge B(a, b)]$$
$$\neg C(a) \wedge [B(a, b) \wedge A(a)]$$
$$A(a) \wedge [B(a, b) \wedge \neg C(a)]$$
$$A(a) \wedge [\neg C(a) \wedge B(b)]$$
$$[B(a, b) \wedge \neg C(a)] \wedge A(a)$$
$$[\neg C(a) \wedge B(a, b)] \wedge A(a)$$

4.3.13 Vyhledávání vzorů podle řetězce složitosti

Při vyhledávání vzorů podle řetězce složitosti využijeme skutečnosti, že řetězec kódujeme pomocí 32-bitového čísla. Tento způsob vyhledávání je velice efektivní a zároveň poměrně přímočarý pro implementaci. Za předpokladu, že již proběhla fáze předgenerování vzorů a máme tak k dispozici datové soubory s vygenerovanými vzory, bude systém vzory vyhledávat tímto způsobem:

1. Při spuštění systém načte vygenerované vzory z datových souborů do paměti, kde je uloží do hashovací tabulky, ve které se adresou (hashem) stane číslo kódující složitost. Na dané adrese bude uložen seznam všech dostupných vzorů s příslušnou složitostí.
2. Jelikož systém ve fázi předgenerování nevytváří vzory, které vyžadují použití kroku eliminace nadbytečných kvantifikátorů, musíme pro všechny vytvořené záznamy z bodu prvního do hashovací tabulky přidat záznamy nové, jejichž adresa se liší pouze v hodnotě bitu kódujícího nutnost použití kroku eliminace nadbytečných kvantifikátorů. Seznam vzorů na takto nově přidaných adresách však ponecháme zatím prázdný.
3. Uživatel v systému zadá řetězec složitosti, ke kterému chce najít vzory. Zadávání řetězce bude realizováno pomocí zaškrtávacích políček (checkboxů) pro jednotlivé položky řetězce složitosti. K takto zadanému řetězci složitosti systém vypočte číslo kódující tento řetězec a přistoupí na adresu, označme ji *a,* v hashovací tabulce vytvořené v předešlých bodech.
4. Pokud adresa *a* v hashovací tabulce neexistuje, přejdeme k odstavci 4.3.8. Pokud je seznam vzorů na adrese *a* prázdný, jedná se vzor vyžadující použití kroku eliminace nadbytečných kvantifikátorů. Přistoupíme tedy v hashovací tabulce na adresu *b*, která se liší od *a* pouze v bitu, odpovídajícímu použití kroku eliminace nadbytečných kvantifikátorů. Pro každý vzor uložený na adrese *b* podle algoritmu popsaného v části 4.3.1.2 vygenerujeme vzor, u kterého bude nutné při skolemizaci použít krok eliminace nadbytečných kvantifikátorů. Všechny takto nově vygenerované vzory uložíme do hashovací tabulky na adresu *a*.

5. Nakonec systém uživateli zobrazí všechny vzory nacházející se v hashovací tabulce na adrese *a* vypočtené v bodu třetím.

Nakonec musíme specifikovat, jak bude systém vyhledávat podle hierarchického řádu vstupních vzorů. Jelikož se pro každý hierarchický řád vytváří samostatný datový soubor, nejjednodušší variantou se jeví vytvářet podobně pro každý hierarchický řád samostatnou hashovací tabulku.

4.3.14 Efektivní vyhledávání vzorů s podobným řetězcem složitosti

Řetězec složitosti obsahuje celkem 23 položek. Jelikož v systému pro generování vzorů nebudeme pracovat s položkou hierarchický řád vzoru po skolemizaci, omezíme počet položek, dle kterých lze vyhledávat, na 22. Dále nebude možné generovat vzory obsahující jiné binární spojky než konjunkce, disjunkce, implikace a ekvivalence. Tímto redukujeme počet položek, podle kterých se bude vyhledávat, na hodnotu 20. Poslední položka řetězce složitosti, tedy hierarchický řád vzoru je čtyř-hodnotová, všechny ostatní jsou dvou-hodnotové. Znamená to tedy, že ve vymezené podmnožině jazyka PL1 máme možnost vygenerovat vzory s nejvýše 2^{21} (= 2 097 152) různými řetězci složitosti.

Pro některé řetězce složitosti z logiky věci nemohou existovat příslušné vzory. Například u vzorů hierarchického řádu 0 a 1 není možné aplikovat distributivní zákony. Stejně tak existují řetězce, ke kterým systém neumí vygenerovat vzory.

Pokud uživatel zadá na vstupu řetězec složitosti, ke kterému systém nemá evidovány žádné vzory, bylo by nežádoucí, aby byl uživatel pouze informován, že systém nenalezl odpovídající vzory v databázi. Mnohem komfortnější a více intuitivní je chování, kdy systém v případě, že nenalezne žádné vzory s řetězcem složitosti na vstupu, vyhledá pro uživatele jiné řetězce složitosti, a to takové, že se liší na nejmenším možném počtu položek, a pro které v databázi vzory existují. Vztaženo k číslu kódujícího řetězec složitosti to znamená, že systém bude hledat čísla, která se liší v nejmenším možném počtu bitů.

Zadá-li uživatel na vstupu řetězec složitosti takový, že pro jeho číselné kódování neexistuje adresa, označme ji *a*, v hashovací tabulce, chování systému bude následující:

1. Systém v hashovací tabulce nalezne takové adresy *b*, které se od *a* liší na nejmenším možném počtu bitů.
2. Systém uživateli přehledně zobrazí řetězce složitosti odpovídající adresám *b,* ve kterých budou barevně zvýrazněny položky, na kterých se příslušný řetězec složitosti liší od řetězce, který uživatel požadoval.
3. Zvolí-li uživatel některý z nabízených řetězců složitosti, systém zobrazí vzory s daným řetězcem složitosti.

5 Závěr

V této práci jsme analyzovali úlohu na skolemizaci formulí PL1 z hlediska obtížnosti pro studenty. Obtížnost úlohy jsme zachytili pomoci řetězce složitosti, u kterého jsme definovali, z jakých položek se skládá, a jaký mají jednotlivé položky význam. Celkový počet položek je 23. Takto vysoký počet položek je způsoben snahou zachytit obtížnost skolemizace co nejdetailněji.

Ve druhé části práce jsme analyzovali, navrhli a implementovali systém, který pro formuli PL1 na vstupu vrátí její řetězec složitosti na výstupu. Systém jsme navrhli tak, že se skládá ze dvou částí. Prvním je lexikální a syntaktický analyzátor jazyka PL1. Druhým je subsystém, který na syntaktický strom vstupní formule uplatňuje postupně jednotlivé kroky skolemizace. Popsali jsme rovněž algoritmy, které přesně určí, jak se úpravy vstupní formule projeví na jednotlivých položkách řetězce složitosti.

V poslední části jsme analyzovali, navrhli a implementovali systém, který pro řetězec složitosti na vstupu vydá vzory formulí PL1 na výstupu. Tento systém rovněž umožňuje vytváření konkrétních úloh ze vzorů. Generování vzorů na základě řetězce složitosti jsme realizovali ve dvou fázích. Nejprve se vzory vygenerují a spočítá se jejich složitost. Následně při požadavku uživatele, se z databáze vzorů vyberou takové vzory se stejným řetězcem složitosti, jaké uživatel požadoval.

Cíle práce byly splněny. Oba systémy, které jsme implementovali, bude možné použít při výuce předmětu Matematická logika, kde budou sloužit jak studentům, tak pedagogům.

Další vývoj obou implementovaných systémů je možný. Zejména u systému pro generování vzorů, se nabízí několik zajímavých a užitečných rozšíření. Například by bylo praktické jej obohatit o funkci generování úloh do nejběžnějších formátů, tedy HTML, PDF, apod.

Literatura

[1] DUŽÍ, Marie. *Matematická logika*. Ostrava: VŠB, Fakulta elektrotechniky a informatiky, Katedra informatiky, 2003.

[2] JANČAR, Petr. *Teoretická informatika*. Ostrava: VŠB, Fakulta elektrotechniky a informatiky, Katedra informatiky, 2007.

[3] BENEŠ, Miroslav. *Překladače*. Ostrava: VŠB, Fakulta elektrotechniky a informatiky, Katedra informatiky, 1993.

[4] ČÍHALOVÁ, Martina a MENŠÍK, Marek. *Metodika hodnocení obtížnosti příkladů řešených v kalkulu přirozené dedukce a obecné rezoluční metody*. Ostrava: VŠB, Fakulta elektrotechniky a informatiky, Katedra informatiky, 2010.

[5] LITSCHMANNOVÁ, Martina. *Vybrané kapitoly z pravděpodobnosti*. Ostrava: VŠB, Fakulta elektrotechniky a informatiky, 2011.

[6] DOXYGEN. *Doxygen manual* [online]. [cit. 2012-04-30].
Dostupné z: http://www.stack.nl/~dimitri/doxygen/

Seznam obrázků

Seznam tabulek

Seznam příloh

A Uživatelská dokumentace

V této příloze uvedeme uživatelskou dokumentaci k oběma implementovaným systémům. Pro názornost budeme do textu vkládat obrázky z grafického rozhraní obou aplikací. Obě aplikace jsou vytvořené jako desktopové. Instalace není nutná, stačí pouze rozbalit soubor "rozbal.zip" z přiloženého CD. Po rozbalení se vytvoří dva adresáře odpovídající implementovaným systémům:

- Systém pro zachycení složitosti - adresář "Složitost skolemizace".
- Systém pro generování vzorů - adresář "Generátor vzorů".

Pro spuštění jednotlivých aplikací, stačí v příslušném adresáři spustit soubor s příponou exe:

- Systém pro zachycení složitosti - "Složitost skolemizace.exe"
- Systém pro generování vzorů - "Generátor vzorů.exe".

Minimální požadavky na konfiguraci a prostředí pro obě aplikace jsou následující:

- Microsoft Windows Vista, 7
- .NET Framework 4.0
- 1GB RAM
- 150MB volného místa na pevném disku.

A.1 Vnitřní abeceda systému

Obě vytvořené aplikace používají vnitřně odlišnou abecedu pro zápis formulí. Vnitřní abeceda je popsána v následující tabulce.

Tabulka 1: Vnitřní abeceda systémů

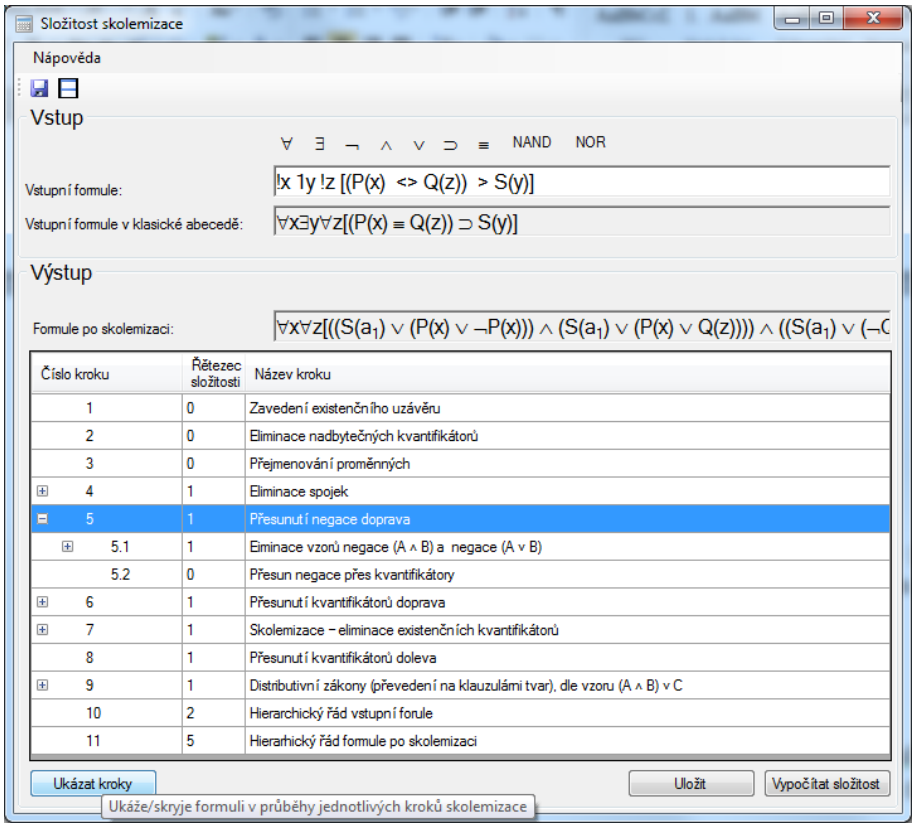
Symbol	**Symbol ve vnitřní abecedě**
všeobecný kvantifikátor	!
existenční kvantifikátor	1
negace	-
konjunkce	*
disjunkce	+
implikace	>
ekvivalence	<>
NAND - negovaná konjunkce	NAND
NOR - negovaná disjunkce	NOR
predikátové symboly	A, B, C, … případně následovány sérií čísel

symbol konstanta	a(), b(), c(), … případně následovány sérií čísel
funkční symboly	a, b, c, … případně následovány sérií čísel
předmětové (individuové) proměnné	x, y, z, … případně následovány sérií čísel
symboly závorek	(, [, {, },],)

Obě aplikace jsou navržené pro intuitivní ovládání. U tlačítek se uživateli zobrazují popisky, které vysvětlují jejich funkcionalitu.

A.2 Dokumentace systému pro zachycení složitosti

Po spuštění aplikace se zobrazí hlavní okno, které je na obrázku níže.



Obrázek 15: Ukázka hlavního okna aplikace "Složitost skolemizace"

A.2.1 Zadávání formule a výpočet řetězce složitosti

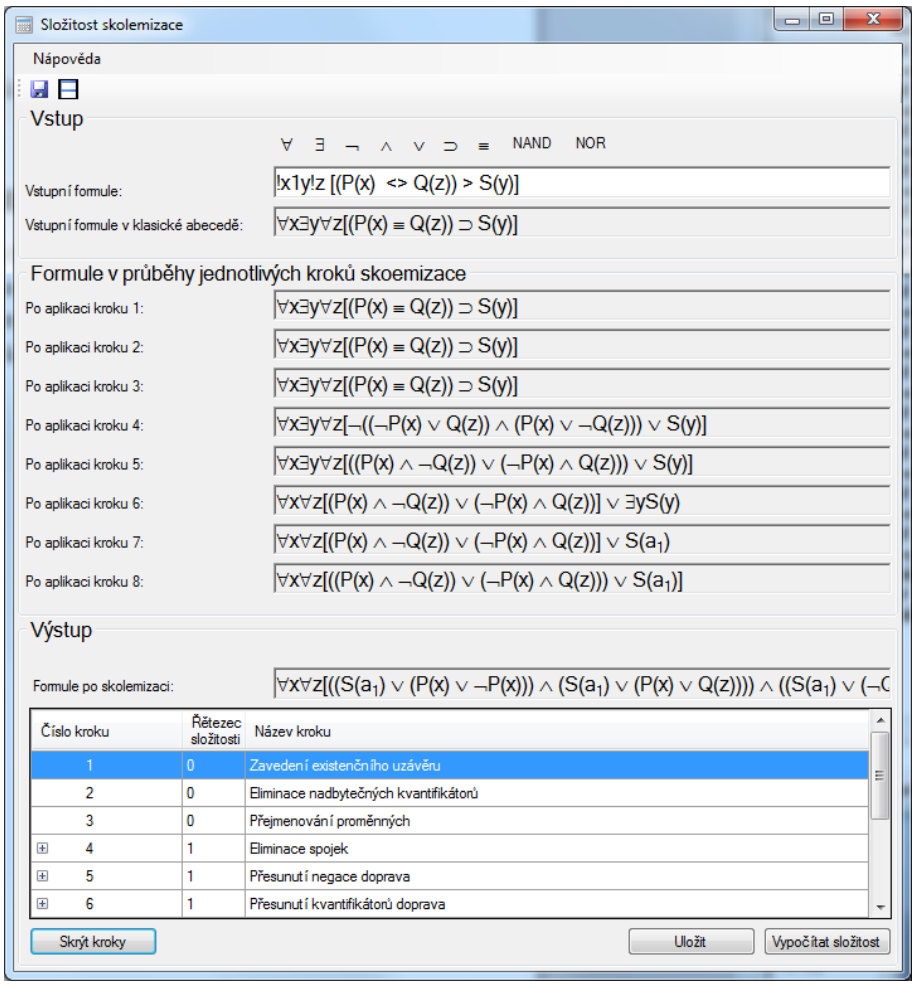
Do textového pole "Vstupní formule" zadáme formuli, jejíž řetězec složitosti chceme vypočítat. Formuli musíme zadávat ve vnitřní abecedě, která je uvedena v tabulce č. 7. Pro jednoduchost můžeme symboly kvantifikátorů a spojek do textového pole zadávat pomocí stisku tlačítek, která

se nacházejí nad textovým polem "Vstupní formule". V textovém poli "Vstupní formule v klasické abecedě" se pro kontrolu zobrazuje vstupní formule v klasické abecedě.

Pro výpočet řetězce složitosti stiskneme tlačítko "Vypočítat složitost" v pravém dolním rohu hlavního okna. Hodnoty složek řetězce složitosti se zobrazí společně s jejich čísly a popisem v tabulce. Složky řetězce složitosti, které se dál dělí, můžeme rozbalit kliknutím na ikonu "+" na řádku dané složky.

A.2.2 Zobrazení průběhu a výsledku skolemizace

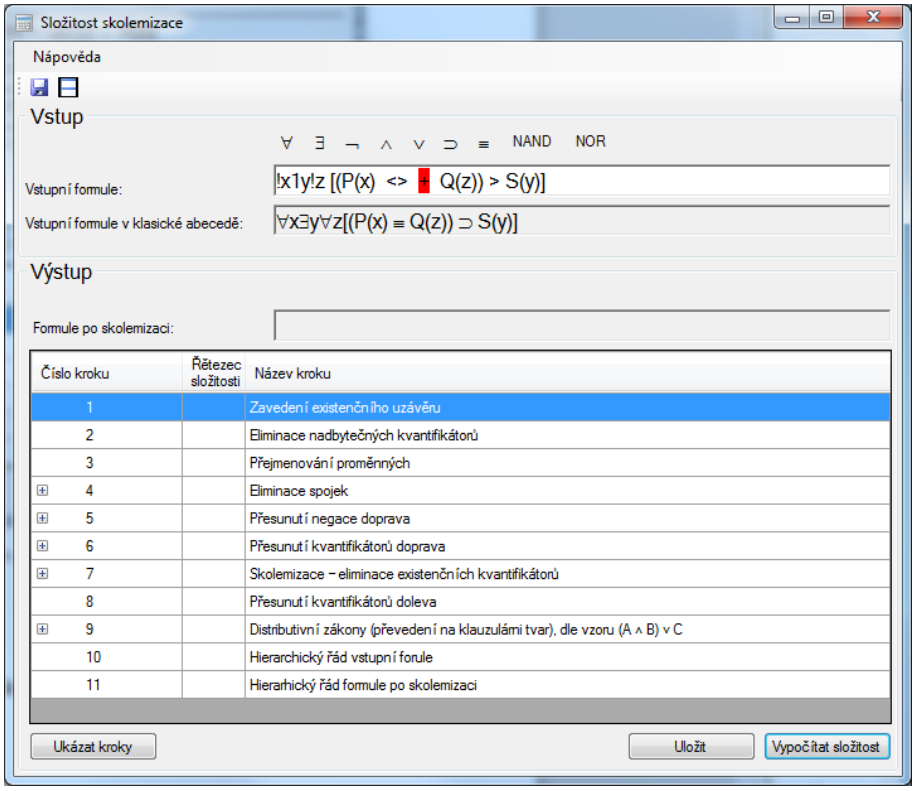
Aplikace zobrazuje také formuli po skolemizaci a rovněž si můžeme zobrazit formuli v průběhu skolemizace. To jest, jak se měnila po aplikaci jednotlivých kroků skolemizace. Výsledná formule je zobrazena v textovém poli "Formule po skolemizaci" a zobrazení formule po aplikaci jednotlivých kroků aktivujeme stiskem tlačítka "Ukázat kroky".



Obrázek 16: Ukázka zobrazení průběhu skolemizace v aplikaci "Složitost skolemizace"

A.2.3 Chybové stavy

Zadáme-li nesprávně utvořenou formuli, aplikace nás upozorní, že se jedná o nesprávně utvořenou formuli a zvýrazní, na které pozici je chyba.



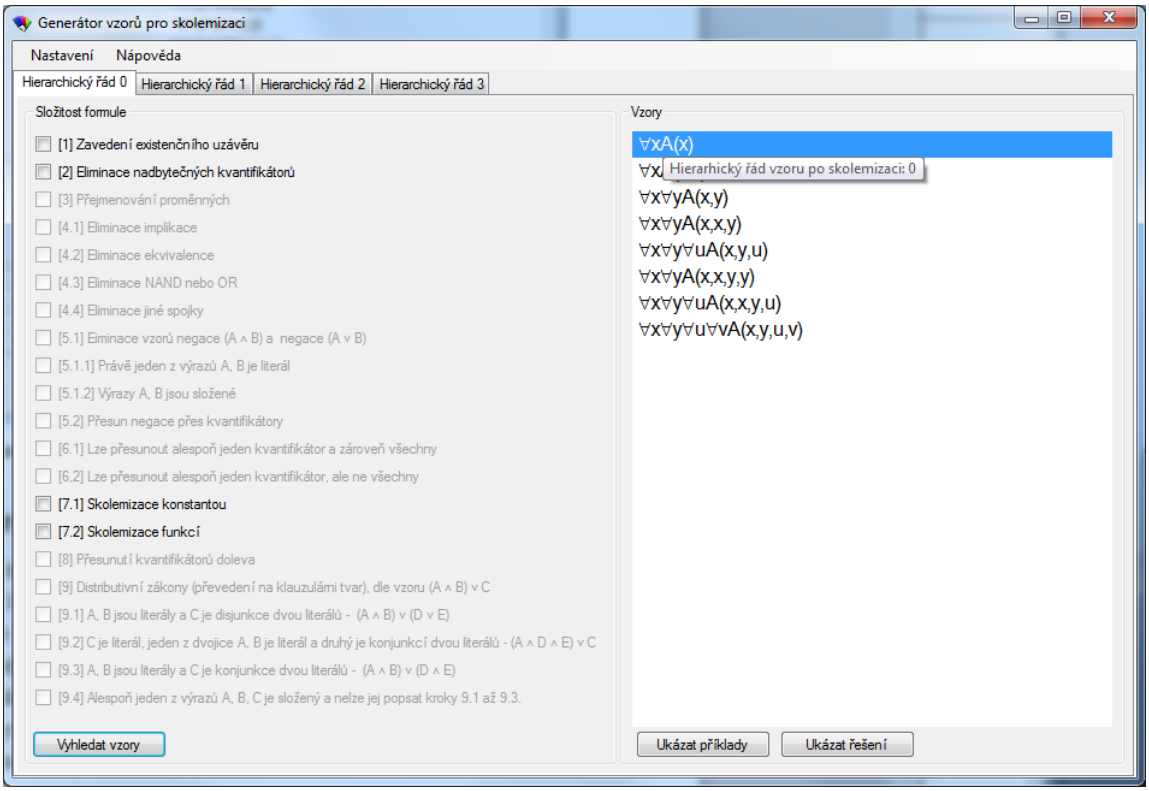
Obrázek 17: Ukázka chybového stavu v aplikaci "Složitost skolemizace"

A.2.4 Další funkce

Aplikace dále umožňuje ukládání vstupních formulí do textového formátu. Pro uložení formule stiskneme tlačítko "Uložit". Vzory se ukládají do souboru "database.txt" v adresáři, ze kterého byla aplikace spuštěna.

A.3 Dokumentace systému pro generování vzorů

Po spuštění aplikace se zobrazí hlavní okno, které je na obrázku níže.

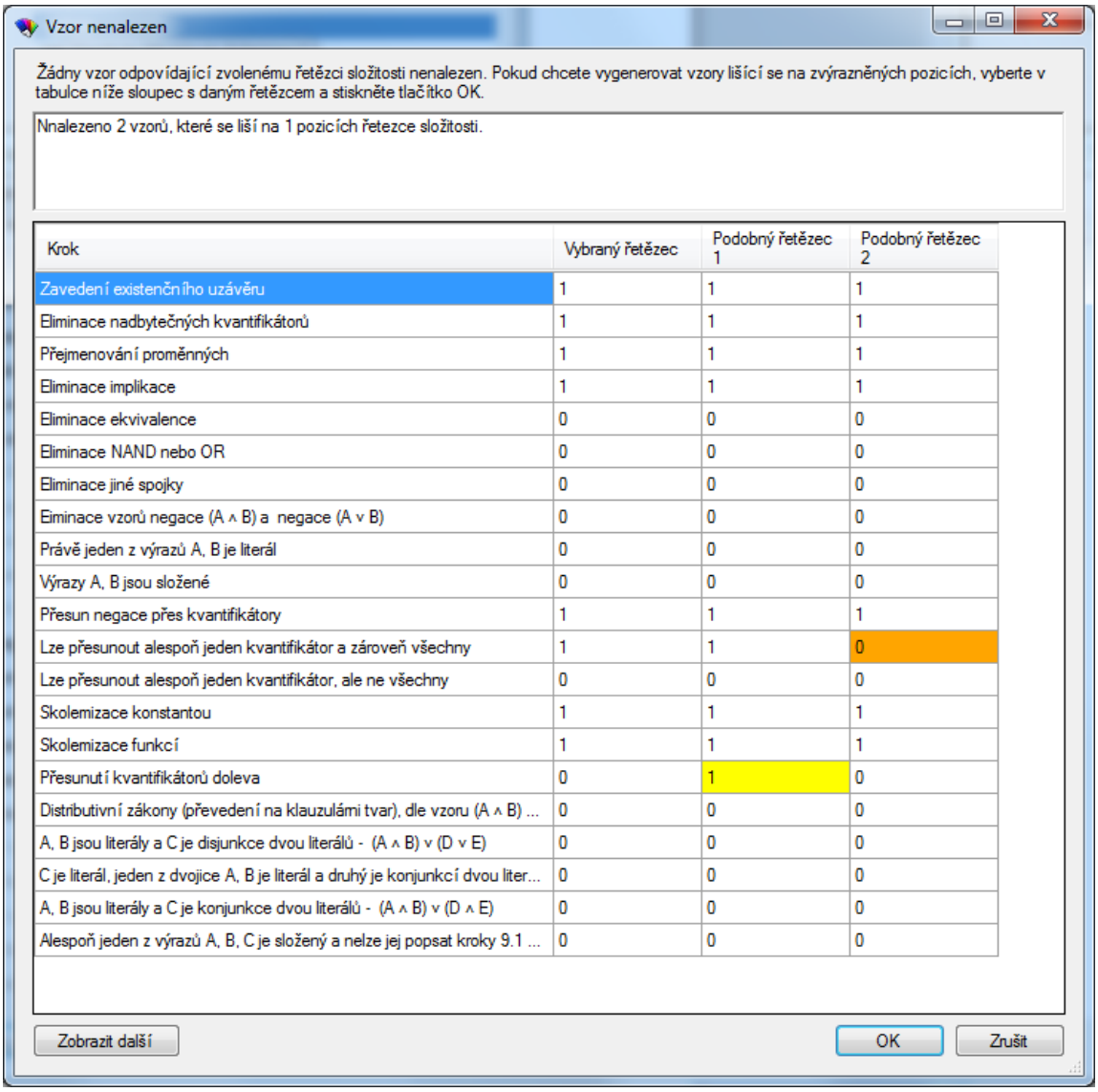


Obrázek 18: Ukázka hlavního okna aplikace "Generátor vzorů"

A.3.1 Nastavení parametrů pro výběr vzorů

Nejprve vybereme hierarchický řád, jehož vzory chceme získat. Hierarchickým řádům odpovídají jednotlivé záložky. Jakmile vybereme hierarchický řád, můžeme pomocí zaškrtávacích políček vybrat, které složky řetězce složitosti mají mít hodnotu jedna, a které hodnotu nula. Zaškrtnuté políčko odpovídá hodnotě jedna na příslušné složce řetězce složitosti. Nakonec stiskneme tlačítko "Vyhledat vzory". V levé části okna se nám zobrazí nalezené vzory. Pokud nad vzorem přejedeme myší, ukáže se nám hodnota hierarchického řádu po jeho převedení do Skolemovy klauzulární formy.

Pokud nebyl nalezen žádný vzor, zobrazí se okno, které je na obrázku níže.

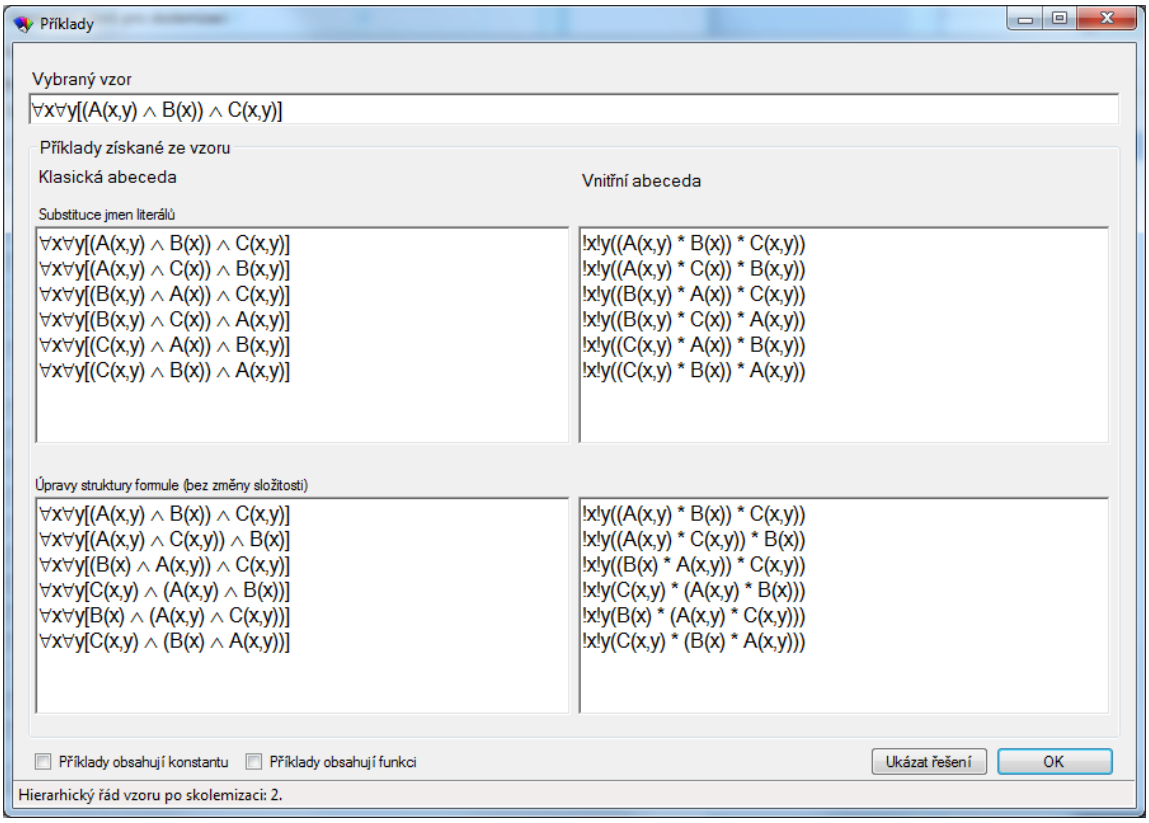


Obrázek 19: Okna zobrazené v případě, že nebyl nalezen žádný vzor

Ve druhém sloupci tabulky se nám zobrazí řetězec složitosti, který jsme chtěli vyhledat. V dalších sloupcích jsou pak zobrazeny podobně řetězce složitosti, pro něž existují v databázi vzory. Složky, na kterých se navržené podobné řetězce složitosti liší od námi zadaného, jsou pro lepší orientaci podbarveny. Stiskem tlačítka "Zobrazit další" můžeme zobrazit více podobných řetězců složitosti. Klikneme-li do sloupce s podobným řetězcem složitosti a stiskneme tlačítko "OK", systém automaticky vyhledá vzory s vybraným podobným řetězcem složitosti.

A.3.2 Generování příkladů

Systém nám umožňuje generovat ze vzorů konkrétní příklady pro skolemizaci. Klikneme-li dvakrát na vyhledaný vzor v hlavním okně aplikace, nebo stiskneme-li tlačítko "Ukázat příklady", zobrazí se nám okno, které je na následujícím obrázku.



Obrázek 20: Okno s jednotlivými příklady na skolemizaci

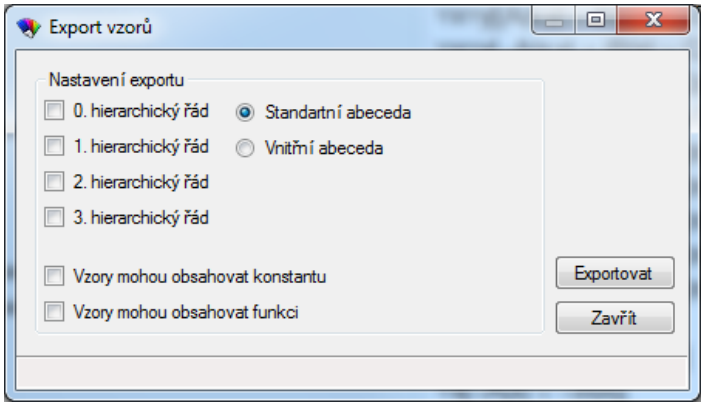
Příklady jsou zobrazeny jak ve vnitřní, tak v normální abecedě. V horní části okna jsou zobrazeny příklady vzniklé pouze substitucí jmen literálů. V dolní části okna jsou pak zobrazeny příklady vzniklé změnou struktury vzoru. Chceme-li, aby příklady obsahovaly funkci nebo konstantu, můžeme zaškrtnout příslušné políčko v levé dolní části okna. Aplikace pak zobrazí formule, ve kterých bude konstanta nebo funkce.

A.3.3 Ukázka řešení vybraného vzoru

Aplikace "Generátor vzorů" spolupracuje s předešlou aplikací, která počítá řetězec složitosti. Stiskneme-li tlačítko "Ukázat řešení" v hlavním okně, nebo v okně s příklady, otevře se nám předešlá aplikace a je nám zobrazen průběh a výsledek skolemizace.

A.3.4 Export vzorů

Aplikace umožňuje export vzorů do textového souboru. Okno pro export vzorů, které je na obrázku níže, zobrazíme v horním menu: Nastavení > Export vzorů.



Obrázek 21: Okno pro export vzorů

Při exportování vzorů můžeme vybrat abecedu, ve které budou vzory uloženy, hierarchické řády, jejichž vzoru budou exportovány a zda chceme generovat i vzory obsahující funkce nebo konstanty. Je vhodné upozornit, že vhledem k počtu vzorů třetího hierarchického řádu může exportování zahrnující třetí hierarchický řád trvat dlouho.

Vzory jsou exportovány společně s jejich řetězcem složitosti. Oddělovačem hodnot jednotlivých složek řetězce složitosti je ",". Ukázku jednoho řádku souboru uvádíme níže.

```
1y!u!q1vA(x,f(y),u,v)
        1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0
```

B Přílohy na CD

Tabulka 2: Adresářová struktura přiloženého CD

Adresář	Popis
\dp	Elektronická verze diplomové práce.
\src	Zdrojové soubory obou implementovaných systémů.
\doc	Programátorská dokumentace k oběma systémům vytvořená pomocí nástroje Doxygen. [6]
\install	Adresář s archivem "rozbal.zip" obsahujícím zkompilované verze obou systémů, které jsou připraveny pro spuštění a použití.